Department of the Army
Pamphlet 73–7

Test and Evaluation

Software Test and Evaluation Guidelines

Headquarters
Department of the Army
Washington, DC
25 July 1997

Unclassified

SUMMARY of CHANGE

DA PAM 73–7
Software Test and Evaluation Guidelines

This new Department of the Army pamphlet--

- o Implements the policies and procedures contained in Department of Defense Directives (DODD) 5000.1, DODD 8000.1, and DODD 5000.2-R and Army Regulation (AR) 25-3 and AR 73-1 (paras 1-2 and 1-4).

- o Provides an overview of the software test and evaluation (T&E) process (chap 3).

- o Details software T&E responsibilities (para 4-3).

- o Describes the software T&E process from pretest activities (chap 5) through software and system testing (chap 6), fielding and transition to maintenance (chap 7), and post deployment software support (chap 9).

- o Provides guidance for integrating software metrics into software T&E and continuous evaluation procedures (chap 10).

Headquarters
Department of the Army
Washington, DC
25 July 1997

Department of the Army
Pamphlet 73–7

Test and Evaluation

Software Test and Evaluation Guidelines

By Order of the Secretary of the Army:

DENNIS J. REIMER
General, United States Army
Chief of Staff

Official:

JOEL B. HUDSON
Administrative Assistant to the Secretary of the Army

History. This publication is a new DA pamphlet. This publication has been reorganized to make it compatible with the Army electronic publishing database. No content has been changed.

Summary. This pamphlet provides guidance and procedures to implement test and evaluation policy for materiel and information systems as promulgated by AR 73–1. It provides detailed guidance for preparing and conducting a test and evaluation program for software–intensive Army systems.

Applicability. The provisions of this pamphlet apply to the Active Army, the Army National Guard, and the U.S. Army Reserve.

Proponent and exception authority. The proponent of this pamphlet is the Deputy Under Secretary of the Army (Operations and Research) (DUSA (OR)). The DUSA (OR) has the authority to approve exceptions to this pamphlet that are consistent with law and controlling regulation. The DUSA (OR) may delegate this authority, in writing, to a division chief within the proponent agency in the grade of colonel or the civilian equivalent.

Suggested Improvements. Users are invited to send comments and suggested improvements on DA Form 2028 (Recommended Changes to Publicatons and Blank Forms) directly to ATTN DACS–TE, TEST AND EVALUATION MANAGEMENT AGENCY, OFFICE OF THE CHIEF OF STAFF, 200 ARMY PENTAGON, WASHINGTON DC 20310–0200.

Distribution. Distribution of this publication is made in accordance with initial distribution number (IDN) 095498, intended for command levels D and E for the Active Army, the Army National Guard, and the U.S. Army Reserve.

Contents (Listed by paragraph and page number)

Unclassified

Contents—Continued

Contents—Continued

Contents—Continued

Figure List

Contents—Continued

RESERVED

Chapter 1
Army Software Test and Evaluation

Section I
General

1–1. Purpose

a. Function. This pamphlet is a guide for implementing software test and evaluation (T&E) and continuous evaluation (CE) policy as prescribed in Army Regulation (AR) 73–1 and other governing Department of Defense (DOD) and Department of the Army (DA) directives.

b. Scope. The material contained herein applies to all Army software T&E performed for systems developed and/or maintained in accordance with the AR 70 and AR 25 series of regulations. This includes T&E associated with development and support of the software aspects of firmware. This pamphlet supports, but does not replace, system–level T&E guidance described in other DA pamphlets of the 73 series and should be used to augment them in the specific area of software.

c. Objective. The objectives of DA Pamphlet (Pam) 73–7 are to—

(1) Present a unified software test and evaluation process for materiel system computer resources (MSCR) and automated information systems (AISs).

(2) Provide implementation procedures for AR 73–1 policies related to software T&E.

(3) Describe a disciplined approach to life cycle software T&E.

(4) Serve as the Army standard for planning and implementing software T&E. This will promote—

(a) Consistency and ease of application.

(b) Early involvement of the T&E community in the acquisition process.

(c) Demonstration of software capabilities.

(d) Acquisition process improvements.

(5) Actively support the principles of total quality management (TQM) and integrated product teams (IPTs).

1–2. References

Required and related publications and prescribed and referenced forms are listed in appendix A.

1–3. Explanation of abbreviations and terms

Abbreviations and special terms used in this pamphlet are explained in the Glossary.

1–4. Policy basis of software T&E

Army software T&E processes and practices have evolved over the years, responding to new technologies, resource constraints, organization changes, and lessons learned. These processes and practices require all Army systems with software, whether MSCR or AIS, to undergo product evaluation throughout their life cycles. Department of Defense policies for MSCR and AIS are described in DOD series 5000 and 8000 regulations. Department of Defense policy provides the basis for Army T&E policy and procedures. The procurement cycle milestones (MSs) and system life–cycle phases described in DA Pam 73–7 are summarized in figure 1–1. Software–specific events and activities are discussed in subsequent chapters. The DOD directives (DODD) and ARs that provide the foundation for software T&E are identified in table 1–1.

1–5. T&E and risk management

As well as describing current Army testing policies, this pamphlet shows the relationship of software products and functions as integral components of their larger systems. Software T&E, CE, incremental working level reviews, increased user involvement in software processes, software metrics, and other strategies are described as means to increase knowledge, awareness, and control of the software development and maintenance process throughout a system's life cycle. This additional insight will assist in highlighting areas of technical risk more uniformly at earlier stages so that they may be addressed more expediently than in the past.

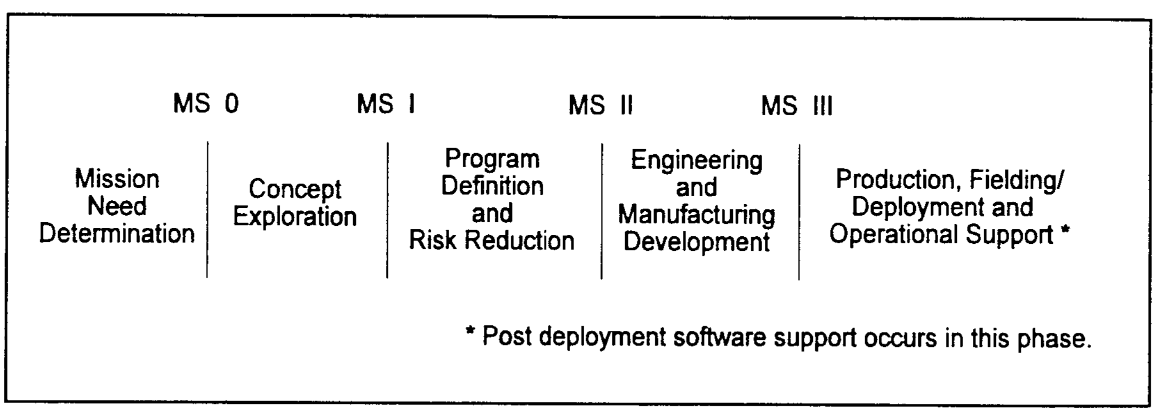


Figure 1-1. System decision milestones and life–cycle phases

Table 1–1
Policy foundation for software T&E

Identification	Title
DODD 5000.1	Defense Acquisition
DOD 5000.2–R	Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs
DODD 8000.1	Defense Information Management Program
AR 70–1	Army Acquisition Policy
AR 73–1	Test and Evaluation Policy
AR 25–3	Army Life Cycle Management of Information Systems

Section II
Continuous Evaluation

1–6. Background

a. AR 73–1 defines CE as " ...a process which provides a continuous flow of T&E information on system status and will be employed on all acquisition programs." Through analysis of available data, CE assesses technical and operational performance, functionality, effectiveness, suitability, maintainability, and supportability and identifies risks. Software CE is performed by personnel involved in software engineering, software quality assurance (SQA), software configuration management (SCM), independent verification and validation (IV&V), Government developmental and operational evaluation, and software developers.

b. Continuous evaluation is fundamental to the proper management of a system. Decisionmaking must be based upon substantive evaluations of software characteristics, and indicators of maturity and reliability throughout the life cycle. These evaluations are paramount to producing quality software which meets user needs. Failure to perform objective evaluations throughout the life cycle has resulted in significant software deficiencies, system delays, and cost overruns.

1–7. Objective of CE

The objective of software CE is to provide impartial assessments of software and system progress. It is a form of risk analysis. Whenever possible, assessments take into consideration the perspective of an operational user.

1–8. Scope of CE

Continuous evaluation begins during the mission need determination and concept exploration phases and continues through post deployment support to system retirement. Software CE activities are tailored to the scope and cost of development or post deployment support and to the criticality of the system's software to its user's mission.

1–9. CE activities and levels of evaluation

a. Continuous evaluation activities are conducted at three levels by—

(1) Individuals in the development organization. The goal is to directly assess and improve quality through development activities. The results of these evaluations are reported within these organizations to document, correct, and reexamine deficiencies. Development organizations are those contractors or Government agencies tasked to perform software development or maintenance for a system.

(2) Individuals associated with the project/program/product manager (PM) such as SQA, SCM, and other matrix support personnel. The goal is to assess software and system quality, conformance to specifications, maturity, reliability, and stability. Evaluations performed within the matrix support organizations are reported to the PM, the developer, and independent testers and evaluators.

(3) Government evaluators or assessors in accordance with AR 73–1. The goal is to formally assess software and system acceptability with respect to operational effectiveness and suitability for deployment and use. Independent evaluators report results to the acquisition community, such as the PM and program executive officers (PEOs), and decisionmaking bodies, such as the Major Automated Information Systems Review Council (MAISRC) or Army Systems Acquisition Review Council (ASARC).

b. Continuous evaluation uses all available data sources and relies on the sharing of that data. It is imperative that the results of all evaluations be shared among the members of the IPT to prevent duplication, ensure deficiencies are identified, and corrective actions are effective. Continuous evaluation consists of activities such as—

(1) Collecting and analyzing data on software engineering processes and procedures in order to identify those which permit poor quality products to be developed. The faulty approaches can then be modified or replaced. Examples of software engineering procedures are requirements analysis, requirements walk–through and reviews, design procedures, quality control procedures, formal reviews and audits, and testing.

(2) Collecting and analyzing data on software products and the results of events in order to determine the status of software/system progress and maturity. Examples of products and events are system/subsystem specifications, system design reviews, code inspections, and tests. Maturity indicators include quality, reliability, stability, and maintainability. The International Organization for Standardization (ISO) 9000 series of standards and the Software Engineering Institute's capability maturity model can be useful when performing these evaluations.

(3) Collecting and analyzing data on corrective actions to ensure proper resolution.

c. The level of CE required for each system throughout its life cycle is determined by its acquisition category, cost, type of dollars expended, and oversight interest.

Section III
Software versus System Testing

1–10. General

a. Unifying the T&E process requires that the software T&E mission include not only software, but its capability to perform operational mission requirements as an integral part of the target system. There are two objectives to testing: demonstration of performance and assisting fault detection and removal procedures.

b. Software T&E must address system level requirements which include, but are not limited to, performance, training, interoperability and interfaces with other systems, supportability, continuity of operations, and user interfaces. These aspects of the total system must be tested and evaluated with the software functions. Examples of specialized system tests which are part of the software integration process are identified in chapter 6.

c. An incremental test strategy allows a variety of test events which are diverse enough to provide confidence in the effectiveness of the test process. In addition, an incremental strategy provides a means to identify and correct failures earlier and more effectively. Figure 1–2 shows the general relationship between different levels of requirements and corresponding test levels. Specific test events and levels are tailored to the needs of each acquisition.

d. Independence in testing and reporting channels promotes objectivity in test and evaluation activities. Army requirements for independent testing are cited in AR 73–1. Independence in reporting is discussed in chapter 4.

1–11. Test levels

a. Software tests. Lower levels of software tests performed by the software developers are structured to verify the accuracy of algorithms and computations and to make sure that the portions of coded software work in accordance with the design, meet the expected results, can handle erroneous inputs, and have been exercised with combinations of differing functions. Software developers are responsible for ensuring that user requirements are correctly implemented in their designs and ensuring that when pieces of software are integrated, they function as required.

b. Software/system tests. As the software subsystems are integrated, software developers ensure that realistic stress and interoperability are verified in tests at systems integration. This is the final opportunity to check software requirements prior to technical tests run by the Government at the system level. Software tests are normally conducted in software development facilities and laboratories.

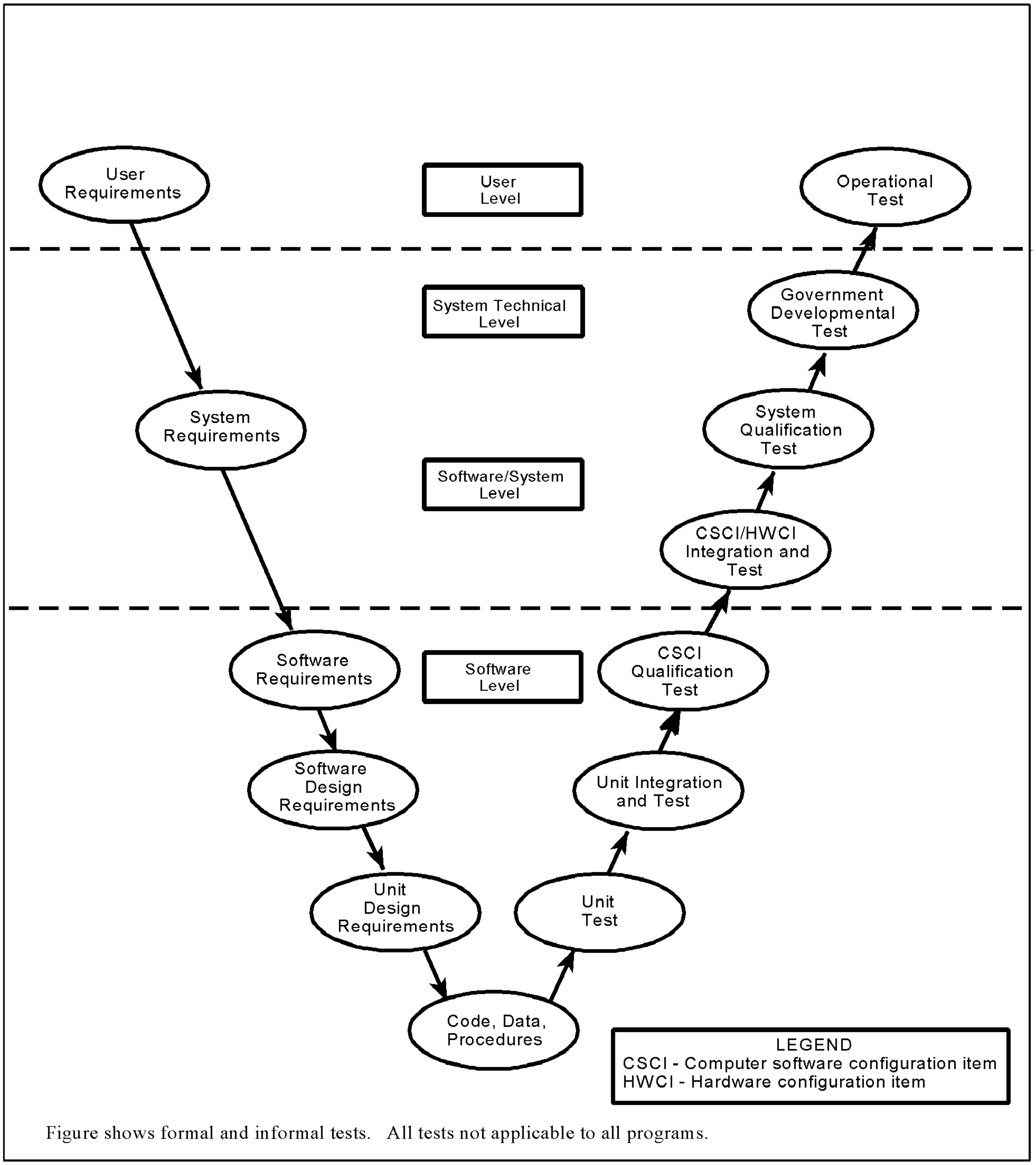


Figure 1-2. Requirements and test level relationship

c. System technical tests. Technical system–level tests look at the capability of the software to support system performance. Government–run technical tests, called developmental tests (DTs) are conducted in the laboratory, Government test beds, and/or user environments using qualified civilians or soldiers. Government developmental testing is structured to subject the system to stress levels commensurate with those that the mature system will be subjected to in representative operating environments. These tests may be structured to estimate the outer limit of the system's operational envelope, if required. Engineering requirements, performance, and user requirements are also examined during DT.

d. Operational tests. Operational testing is system–level testing which focuses on effectiveness, suitability, and survivability. Characteristically, operational tests involve the intended user troop units or organizations and take place in realistic operational environments. Operational tests may be performed at any time during the life cycle. The need for these tests is determined by the IPT and the acquisition strategy.

Section IV
Organization and Approach

1–12. Pamphlet organization
Following this introduction, DA Pam 73–7 is arranged as follows:

a. Chapter 2 supplies a common basis for terms used throughout the pamphlet.

b. Chapter 3 outlines Army software test program requirements and issues to consider when planning or evaluating a test program.

c. Chapter 4 identifies principal organizations in the software and system acquisition process and their respective roles and responsibilities in T&E.

d. Chapters 5 through 8 provide general procedures for test related or CE tasks typical to major system and software development activities. Tasks performed by personnel in chapter 4 who are not members of the developer's organization are included. Each activity is presented in the following manner:

(1) *General.* What the activity comprises.

(2) *Objective.* The primary purpose of the activity.

(3) *Entry criteria.* Other activities, products or events that should have occurred prior to one or more steps in the activity under discussion.

(4) *Test activities.* Representative actions relating to planning, executing, or reporting tests that are appropriate for the development activity.

(5) *Evaluation activities.* Representative actions relating to CE that are appropriate for the development activity.

(6) *Metrics.* Software metrics for which data can typically be collected or which are analyzed during the activity.

(7) *Decision criteria.* Summary of test or evaluation outcomes expected to be addressed prior to the end of the activity.

e. Chapter 9 discusses unique post deployment software support considerations relative to the activities in chapters 5 through 8.

f. Chapter 10 supplies detailed guidance on software metrics that support the processes described in this pamphlet.

1–13. Approach

a. Military Standard 498 (MIL–STD–498), the current military standard for software development and documentation acquisitions, and IEEE Std P1498/EIA IS 640, its commercial equivalent, serve as the bases for the activity descriptions. These activities, to varying degrees, are inherent in any software–intensive system. Substituting other commercial or industry standards for software development documentation or other military guidance referenced in the procedures of this pamphlet can be effectively accommodated and is encouraged.

b. This pamphlet does not provide guidance on implementing DOD acquisition reform policy and does not identify alternative commercial substitutes for previously mandated military standards and specifications regarding software development production and interrelated processes. Additional sources for information in this area include, but are not limited to, the DOD Index of Specifications and Standards (DODISS), International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), American National Standards Institute (ANSI), Institute of Electrical and Electronic Engineers (IEEE), and Software Engineering Institute (SEI). The guidance documents referenced in this pamphlet were in effect at the time of publication.

c. This pamphlet does not promote any particular software or system development strategy. It is up to the Government program management and its developer(s) to negotiate appropriate activities, work items within activities, data products, and sequencing of events.

d. Activities are not stand–alone entities, but interact with products and work performed in other activities. They may be dynamic and iterative. Steps in one activity initiate work in other activities. Independent management action can also initiate activities. In general, an activity is comprised of the following elements:

(1) Planning the work to be accomplished in an activity.

(2) Performing the work by executing the plan.

(3) Evaluating (or monitoring) work performed against expected results. Based on the results, replanning often occurs and the cycle repeats.

e. The interconnections among project activities can be viewed as a network. A highly simplified generic integrated project network is shown in figure 1–3. In the figure, activities A, C and E are initiated by program management action at their designated starting times. Evaluation steps in activity A initiate planning for B and execution steps in activity C. Activity G follows an iteration of C.

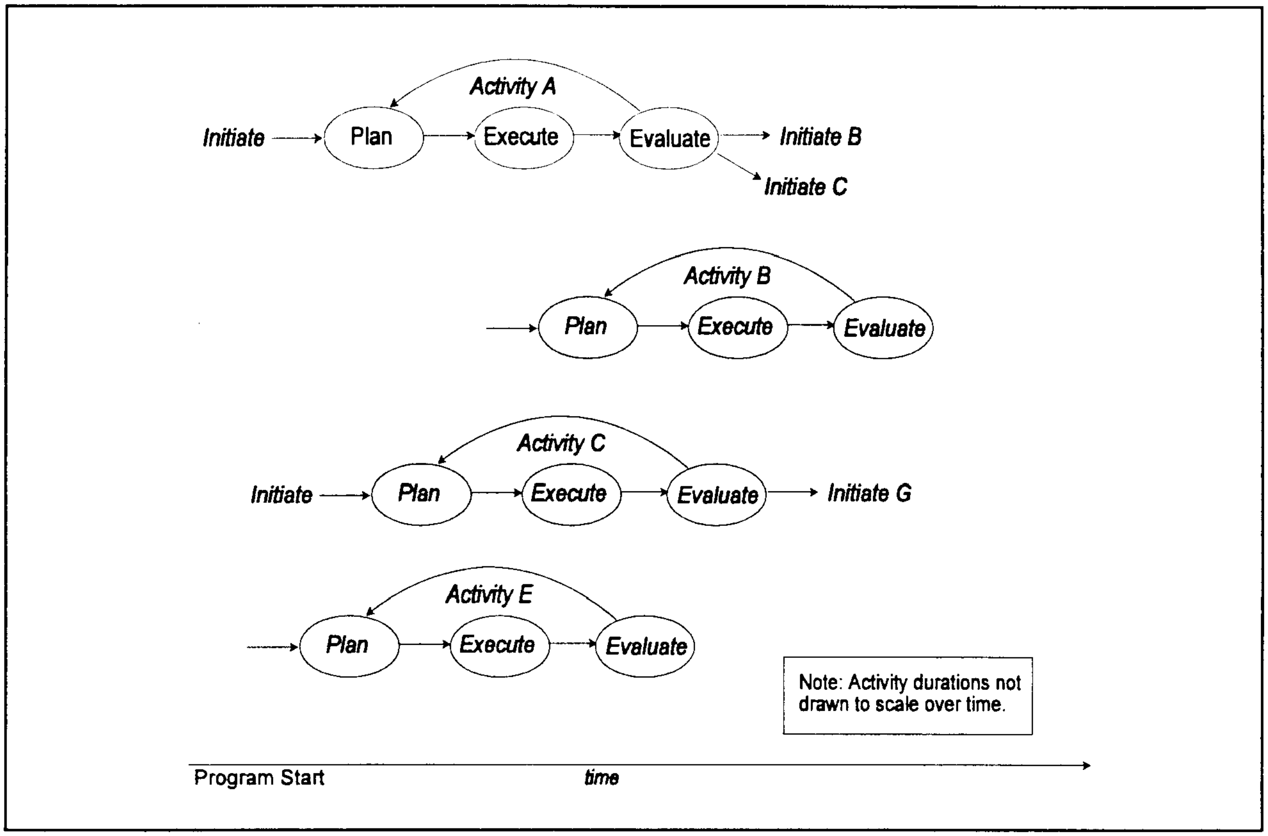


Figure 1-3. Sample integrated project activity network

Chapter 2
Terms and Definitions

2–1. General

a. Key to implementing the software T&E process is the use of consistent and common terminology. This chapter establishes the set of terms used throughout DA Pam 73–7 and references similar terms often used within the AIS and MSCR environments. Acronyms and definitions for many of the terms are contained in the Glossary.

b. Of special note is the term " software developer." Throughout this pamphlet, a developer is any contractor or Government agency that generates software products whether by means of new development, modification, reuse, re–engineering, maintenance or other activity.

2–2. Terminology cross reference

a. Entries under the heading " Current Term" in the following tables describe the T&E process in this pamphlet. Other terms shown are for reference only. Note — In this pamphlet, the term MSCR is equivalent to the computer resources comprising weapons systems, battlefield automated systems (BAS) or mission critical computer resources (MCCR). AIS is equivalent to Army information mission area (IMA) systems.

b. A common set of terms is used to provide a singular approach to describing tests across the Army. Tables 2–1 and 2–2 list the current terms and those which have been used in the past.

c. Organizational roles are presented in table 2–3 to show the relationship between terms currently in use and any deviations in terms used in this pamphlet.

d. In the past, MSCR and AIS programs chose to use a variety of formats for their technical data. These were consolidated into a single set under MIL–STD–498 and Institute of Electrical and Electronics Engineers Standard (IEEE STD) P1498/EIA IS 640 (commercial version) for any type of software acquisition. Table 2–4 shows the consolidated documents and comparable MSCR and AIS software documents of earlier military standards. Systems contracted for prior to MIL–STD–498, of course, may continue to maintain software deliverable data in their original formats. The table also includes documents pertaining to Government DT, operational test, and independent evaluation. Two of the documents have expanded their software role as a result of AR 73–1. They are the users' functional description (UFD) and metrics reports.

Note. MIL–STD–498 and IEEE Std P1498/EIA IS 640 will hereafter be referred to as " consolidated software standards" throughout the rest of this pamphlet.

e. Table 2–5 is a cross reference of terms related to software development tools and techniques.

f. The priorities and categories in tables 2–6 and 2–7 are recommended in order to provide a standard means to score software deficiencies and change requests during development and test. These common classifications form the basis for controlling entry and exit objectives, promoting standard interpretation of test deficiencies, and fostering meaningful entries in the Army test incident reporting system (ATIRS). These priorities and categories were chosen because they incorporate a user's view of problem type and criticality; many other classifications are possible.

Table 2–1
Developmental test terminology cross–reference

Current term	AIS term	MSCR term	Attributes
Government developmental test (DT) [1]	Software qualification test (SQT)	DT, comparison test (CPT), first–article test (FAT), preproduction qualification test (PPQT), technical feasibility testing (TFT) [2]	Formal test(s) requiring independent testers, independent evaluators/assessors. Requires complete design limit tests (stress volume). Required throughout life cycle. Independent evaluation/assessment after MS III (only if new or unresolved issues).
System qualification test	Software development test (SDT) cycle/system test	System integration test (SIT)	Formal test(s) witnessed by Government for system acceptance. Test plans, procedures, conditions prepared by developers. Levels of test, documentation defined by statement of work (SOW). Software metrics collected.
Computer software configuration item (CSCI) qualification test	SDT cycle/system test	CSCI formal qualification test (FQT)	Formal test(s) witnessed by Government for software acceptance. Test plans, procedures, conditions prepared by developers. Levels of test, documentation defined by SOW. Software metrics collected.
CSCI/hardware configuration item (HWCI) integration test	SDT cycle/system testing [3]	System integration testing [3]	Informal tests controlled and performed by software and system developer. Tests and environment documented in software development files (SDF)s. Includes demonstrations or limited functional capability for users to interact with during design or development.
Unit integration test	SDT module/program testing	Computer software component (CSC) integration test	Same attributes as CSCI/HWCI integration test.
Unit tests	SDT unit tests	Computer software unit (CSU) tests	Same attributes as CSCI/HWCI integration test.
Experimental	Prototyping	Prototyping/emulation/ brassboard	Informal tests controlled and performed by developer typically to demonstrate limited functional capability for user feedback during development.

Notes:
[1] See AR 73–1 for test types.
[2] Comparison test (CPT), first–article test (FAT), preproduction qualification test (PPQT), technical feasibility test (TFT).
[3] Prior to formal qualification test.

Table 2–2
Operational test terminology cross–reference

Current term	AIS term	MSCR term	Attributes
Initial operational test (IOT) [1]	Software acceptance test (SAT) [1]	IOT, Joint test (JT), multi––Service operational test (MOT), operational testing (OT) [2]	Required for MS III. Independent testers, independent evaluators/assessors. Class VI AIS systems require IOT/follow–on operational test (FOT), but tester performs evaluation/assessment.
Follow–on operational test (FOT)	SAT [1]	FOT, on–site user test (OSUT) [3]	Independent testers, independent evaluators/assessors only if unanswered issues, new issues, fixes from IOT. All systems with upgrades, fixes, enhancements after MS III.
Supplemental site test (SST)	SST	None	Supplements AIS IOT or user acceptance test (UAT) to exercise all configurations used operationally.
Early user test (EUT), early user experiment (EUE)	Beta or prototype testing (informal)	Early user test and experimentation (EUTE)	To answer issues prior to MS II or identify system solutions and/or define issues at MS II or beyond.
Force development test (FDT), force development experiment (FDE)	Beta or prototype testing (informal)	Force development test and experimentation (FDTE)	Conducted with users under field conditions. Supports acquisition and development of requirements, doctrine, training.
Limited user test (LUT)	None	None	Generally single issue user test between MS II and MS III.
User acceptance test (UAT) [1]	None	None	Post MS III AIS user test when full FOT not needed.
Emergency fixes	Lead site verification test	Tests for emergency fixes	

Notes:
[1] With SST, if necessary.
[2] Joint test (JT), multi–Service operational test (MOT), operational test (OT).
[3] On–site user test (OSUT).
[4] All tests are formal unless otherwise noted.

Table 2–3
Organizational roles cross–reference

Current term	AIS term	MSCR term
Acquirer	User Group	Contracting Agency
Approval Authority	Approval Authority	Program Decision Authority
Central Design Activity (CDA) AIS (no change to MSCR terms)	Software Development Center (SDC)	Life Cycle Software Engineering Center (LCSEC)/ Software Support Activity (SSA)
Computer Resources Integrated Product Team (IPT)	Computer Resources Working Group (CRWG)	CRWG
Continuous Evaluation	Quality Assurance/IV&V	Developmental Evaluators: IV&V/SQA/LCSEC
Developer	Application System Developer/Development Group	Developing Agency/Contractor/Software Engineers
Developmental Tester	Independent Third Party Tester	Developmental Tester IV&V/LCSEC/SQA
FP/CBTDEV or User Representative	Functional Proponent (FP)	Combat Developer (CBTDEV)
Independent Developmental Evaluator/Assessor	Independent Evaluator	Developmental Evaluator/Assessor
Independent Evaluator	Independent Evaluator	Operational Evaluator
Materiel Developer (MATDEV)	Assigned Responsible Agency	Materiel Developer
—	Major Automated Information System Review Council (MAISRC)	Army Systems Acquisition Review Council (ASARC)
--	Non-Automated Data Processing (ADP) Operator, Organization	User/Troops/Unit
Operational Tester	Independent Third Party Tester	Operational Tester
Project/Program/Product Manager (PM)	Project Officer/Operations Manager/Program Manager/Product Manager/Project Manager	Program Manager/Product Manager/System Manager
Software or Developer's Testers	Software Testers	Software Engineering Testers
Software or Developer's Tester	Tester	Developmental Tester
Software Quality Assurance (SQA) Organization	Automation Quality Organization	Assurance (SQA)
T&E Team/T&E Community	Matrix Support and Testers	Matrix Support, PDSS Personnel, Testers, and Independent Evaluators/Assessors
Test IPT	Test Integration Working Group (TIWG)	TIWG
User Representative	Proponent Agency (PA)	Similar to CBTDEV/FP

Notes:
-- AIS/MSCR term still applies.

Table 2–4
Documentation cross-reference

Current term	AIS term	MSCR term
Computer Operation Manual (COM) [1]		Computer System Operator's Manual (CSOM) [3]
Computer Programming Manual (CPM) [1]		Software Programmer's Manual (SPM) [3]
Computer Resources Life Cycle Management Plan (CRLCMP)	Management Plan (software level), System Decision Paper	Computer Resources Management Plan (CRMP)
Database Design Description (DBDD) [1]	Database Specification (DS) [2]	
Event Design Plan (EDP) (operational)	SAT Test Plan	Test Design Plan (TDP) (operational)
Firmware Support Manual (FSM) [1]		Firmware Support Manual (FSM) [3]
Interface Design Description (IDD) [1]	Software Unit Specification (US), interface design info [2]	Interface Design Document (IDD) [3]
Interface Requirements Specification (IRS) [1]	Software Unit Specification (US), interface req't info [2]	Interface Requirements Specification (IRS) [3]
Metrics Reports		Metrics Charts
--	Management Plan	Program Management Plan
--	Software Change Package (SCP)	Engineering Change Proposal - Software
--	Technical Test Plan (formerly SQT, PT)	Detailed Test Plan (DTP)
--	Implementation Procedures (IP) [2]	Life Cycle Software Support Environment User's Guide
--	Problem Reports (PR)	Software Trouble Report (STR)/Software Problem Change Report (SPCR) or Test Incident Reports (TIRs)
--	Maintenance Manual (MM) and Implementation Proc. (IP)	Software Support Transition Plan
--		Developmental Software Support Environment
--		Documentation of Commercially Available/ Privately Developed Software
--	Engineering Change Proposal - Software (ECP-S)	Software Change Notice, Engineering Change Proposal (ECP)
Operational Concept Description (OCD) [1]	Functional Description (FD), section 2 [2]	System/Segment Design Document (SSDD), section 3 [3]

Table 2–4
Documentation cross-reference—Continued

Current term	AIS term	MSCR term
Software Design Description (SDD) [1]	Software Unit Specification (US), design info [2] Maintenance Manual (MM), " as-built" design info [2]	Software Design Document (SDD) [3]
Software Development Plan (SDP) [1]	Functional Description (FD), section 7 [2]	Software Development Plan (SDP) [3]
Software Input/Output Manual (SIOM) [1]	User Manual (UM) [2]	
Software Installation Plan (SIP) [1]	Implementation Procedures (IP) [2]	
Software Problem/Change Report (PCR)	Problem Report, Trouble Report	Software Trouble Report (STR), Problem Report
Software Product Specification (SPS) [1]	Maintenance Manual (MM), maintenance procedures [2]	Software Product Specifications (SPS); [3] Computer Resources Integrated Support Document (CRISD), modification procedures [3]
Software Requirements Specification (SRS) [1]	Software Unit Specification (US), req't info [2]	Software Requirements Specification (SRS) [3]
Software Test Description (STD) [1]	Test Plan (PT), detailed info [2]	Software Test Description (STD) [3]
Software Test Plan (STP) [1]	Test Plan (PT), high level info [2]	Software Test Plan (STP) [3]
Software Test Report (STR) [1]	Test Analysis Report (RT) [2]	Software Test Report (STR) [3]
Software Transition Plan (STrP) [1]	Maintenance Manual (MM), planning info [2]	Support Document (CRISD), planning info [3]
Software User Manual (SUM) [1]	End User Manual (EM) [2]	Software User's Manual (SUM); [3] Training Manuals (for CBTDEV)
Software Version Description (SVD) [1]		Version Description Document (VDD) [3]
Software Quality Program Plan (SQPP)	Automation Quality Plan (AQP)	Software Quality Program Plan (SQPP)
System Evaluation Plan (SEP) (operational)	Independent Evaluation Plan (operational)	Test and Evaluation Plan (TEP) (operational)
System/Subsystem Design Description (SSDD) [1]	System/Subsystem Specification (SS), system design info [2]	System/Segment Design Document (SSDD) [3]
System/Subsystem Specification (SSS) [1]	System/Subsystem Specification and Functional Description, system req't info [2]	System/Segment Specification [3]
Users' Functional Description (UFD) and Functional Description (FD)	Functional Description (FD) [2]	

Notes:
-- AIS/MSCR term still applies.
[1] Document from MIL-STD-498/IEEE std P1498/EIA IS 640.
[2] Document from DOD-STD-7935A.
[3] Document from DOD-STD-2167A.

Table 2–5
Tools and techniques cross-reference

Current term	AIS term	MSCR term
Development Tools	Development Tools, Toolbox	Development Tools, Toolbox
Instrumentation, (Drivers, Emulators, Stimulators) Performance Monitors	Performance Monitors	Drivers, Emulators, Stimulators
Recovery/Reconfiguration Testing	Checkpoint/Recovery Testing	Recovery/Reconfiguration Testing
Software Development File (SDF)	Program Folder	Software Development Folder
Software Development Library	Development Library (DEVLIB)	Software Development Library
Software Engineering Environment	Developer's Environment	Host Environment (Software Engineering Environment)
Software Reliability	Reliability	Software Reliability

Table 2–6
Software problem/change priorities

Current term	Applies if problem could—	AIS term	MSCR term
Priority 1	a. Prevent the accomplishment of an essential capability. b. Jeopardize safety, security, or other requirement designated " critical."	Emergency	Priority 1
Priority 2	a. Adversely affect the accomplishment of an operational- or mission-essential capability, and no work-around solution is known. b. Adversely affect technical, cost, or schedule risks to the project or to life-cycle support of the system, and no work-around solution is known.	Urgent	Priority 2
Priority 3	a. Adversely affect the accomplishment of an operational- or mission-essential capability, but a work-around solution is known. b. Adversely affect technical, cost, or schedule risks to the system or to life cycle support of the system, but a work-around solution is known.	Urgent	Priority 3

Table 2–6
Software problem/change priorities—Continued

Current term	Applies if problem could—	AIS term	MSCR term
Priority 4	a. Result in user/operator inconvenience or annoyance but does not affect a required operational- or mission-essential capability. b. Result in inconvenience or annoyance for development or support personnel but does not prevent the accomplishment of those responsibilities.	Routine	Priority 4
Priority 5	Any other effect.	Routine	Priority 5

Table 2–7
Software problem/change categories

Current term	Product affected	AIS term	MSCR term
Code	The software code	Technical	Software
Database/data file	A database or data file	Technical	Software
Design	The design of the system or software	Technical or functional	Design
Manuals	The user, operator, or maintenance manuals	Documentation	Documentation
Operational concept	The operational concept	Documentation	Documentation
Requirements	The system or software requirements	Documentation	Documentation
Test information	Test plans, test descriptions, or test reports	Documentation	Documentation
Plans	One of the plans developed for the project	Documentation	Documentation
Other	Other software products		

Chapter 3
T&E as Part of Acquisition and Development

Section I
Basis

3–1. General

a. The procedures in this pamphlet present an iterative, structured, and comprehensive approach to software test and evaluation throughout a system's life cycle. Specific application of these procedures should be tailored to the technical and management characteristics of each system acquisition program.

b. Selection and tailoring of software T&E procedures are primarily determined by the level of technical risk which can be allowed in the system acquisition. Other significant program factors, such as urgency to field, may also contribute to tailoring decisions.

c. This chapter discusses general strategies to consider in providing T&E which is responsive to program needs and user requirements and results in fielding quality systems.

3–2. Software T&E program requirements

According to AR 73–1 and AR 380–19—

a. Software T&E must be accomplished within the context of the overall system development and test program. It supports the concept of TQM for the system development. In accordance with the TQM concept, all persons involved in the software development process are responsible for impacting the quality of the software product. The following general guidelines constitute the minimum requirements for a software T&E program.

b. A software T&E program must reflect a systematic and measurable process in which continuous software evaluations present a realistic and iterative status report. Clearly defined risk assessment criteria for each life-cycle phase, metrics, and CE are the basis for a logical progression of software T&E. This progression is based on demonstrating achievement of objectives at each step.

c. Software must be assessed for its ability to support system effectiveness and suitability. Software T&E must reduce risk to an acceptable level that ensures system requirements and mission objectives will not be impaired by deficiencies attributable to software. The user is the ultimate arbitrator of system requirements and mission objectives and must be involved with the T&E program.

d. Software T&E must provide data to support qualitative and quantitative software metrics. These metrics serve as measures and indicators of the critical technical and operational characteristics that both the software and the integrated system need to achieve.

e. A software T&E program must support the IPT approach by effectively sharing T&E results among participating organizations across all life-cycle phases. Each element of the software T&E program must provide data to support software and system acquisition decisions.

f. Software T&E must support the acquisition management process. Individual programs and acquisition strategies determine the scope of a software T&E program.

g. All AIS and MSCR systems which contain classified or sensitive unclassified information must incorporate safeguards to protect against compromise, subversion, or unauthorized manipulation. Formal accreditation by a designated accreditation authority (DAA) is required prior to fielding. Certification, a technical evaluation of security functions that support the mode of operation and security policy for a system, supports accreditation.

h. Use of modeling or simulation in T&E to enhance evaluations and reduce costs is encouraged. However, modeling and simulation may only supplement tests, not replace them.

Section II
Test Program Considerations

3–3. The acquisition strategy

a. For any system, software testers and evaluators need to approach and plan their strategies based on fundamental acquisition characteristics. These include but are not limited to—

(1) The acquisition category.
(2) System development approach.
(3) Software development approach.
(4) The system's complexity.
(5) Deployment philosophy.
(6) Other participating organizations.

b. Factors such as acquisition category and DOD oversight interest determine system-level reporting and approval requirements, and the requirement for independent evaluation.

c. Table 3–1 contains several examples of alternative acquisition program strategies. A system and its software need not share the same strategy or the same degree of iteration in the case of multiple build/block implementations. It is beyond the scope of this pamphlet to offer guidance in determining an appropriate system or software development strategy; the consolidated software standards identified in paragraph 2–2 *d* offer guidance in this area. The selected strategy does, however, significantly affect what technical information is available at what time, and the degree of risk that can be detected or corrected at any point.

d. The system and software activities outlined in chapters 5 through 9 of this pamphlet, taken in aggregate and performed end-to-end, approximate a grand design program strategy. Tailoring to combine activities when feasible, eliminate those not needed, or reflect the needs of particular programs is left to the discretion of the appropriate functional area authority. Examples of tailoring include combined computer software configuration item (CSCI)/system qualification tests, combined or concurrent DT/OT events, and use of nondevelopment items (NDIs). The accelerated software development process (ASDP) approach for AIS is a variant of the incremental strategy (see DA Pam 73–1 and 73–5).

e. Highly complex or wide-area systems may require extensive instrumentation and data reduction capability during large scale Government tests, which must be adequately planned and scheduled.

f. Deployment philosophy considerations could be site specific operational requirements that necessitate multiple field tests.

g. The most efficient and effective use of resources is to coordinate activities among the members of the IPT.

3–4. The type of system

a. MSCR refers to computer resources acquired as integral elements of systems used by military personnel to carry out combat missions. They can be physical components of weapons systems, or computer resources essential to a weapons system's operation and maintenance in the field. The term also applies to ancillary computer resources (hardware, software, documentation, data, and so forth) associated with testing and maintaining the MSCR, such as—

(1) Training devices.

(2) Automatic test equipment.

(3) Land-based test sites.

(4) System integration and test environments.

b. Software T&E of MSCR should address unique characteristics such as real-time processing constraints, security safeguards, fault tolerance, human health and safety concerns, and adverse operating environments.

c. Automated information systems encompass the functions, resources, equipment, software and activities associated with one or more of the disciplines of automation, telecommunications, visual information, publications and printing, and records management. They are most often sustaining base systems.

3–5. Target environment and intended end use

a. Single-site systems are those that reside in only one location. Special T&E considerations include arranging for concurrent live operation during testing periods and using realistic testing methods. An example of a single-site system is an Army wholesale logistics system which resides on a large mainframe at one location. Test designs for single-site systems should account for—

(1) Loading and running the system in specialized test regions which are partitioned from production regions.

(2) Running the volumes and stress loading necessary to simulate or stimulate expected interactions with users, other systems or the external environment. This may require the use of models.

(3) Ensuring that all processing and reporting cycles are executed, whether by models or actual testing, including end-of-year and other cycle roll-ups.

b. Single-user systems are designed to be employed by one user organization. This arrangement allows direct and consistent user involvement throughout development and T&E. It is essential that operational users be part of T&E planning to ensure that specialized operations are not overlooked. Involving users is equally important for all other system types as well.

c. Sustaining base information systems provide the capability to raise, organize, train, equip, deploy, and sustain Army forces. They usually do not physically move to the battlefield during mobilization or wartime. These systems are typically found at centrally located sites. Test designs for these systems should account for—

(1) Performing different missions during peacetime, mobilization, wartime, and demobilization operating conditions.

(2) Demonstrating ability for immediate readiness when necessary.

(3) Demonstrating ability to transition smoothly and rapidly from one operating condition to another.

d. Strategic information systems are typically those which facilitate command and control of Army forces and resources, including planning, directing, controlling, reporting, and communications. Test designs for these large scale systems should account for the same considerations as sustaining base systems with the added emphasis on fail-safe operations and information security. Strategic information systems are most often managed under DOD and Joint Chiefs of Staff (JCS) publications and AR 525-1; however, some may be developed under AR 25-3 policies. An example of a strategic system is the Global Command and Control System (GCCS).

Table 3–1
Examples of program strategies

Program strategy	Define all first requirements first?	Multiple development cycles?	Field interim system/ software?
Grand Design (Once-Through) — Determine user needs, define requirements, design the system, implement the system, test, fix and deliver.	Yes	No	No
Incremental (Preplanned Product Improvement) — Determine user needs, define requirements, plan sequence of builds to implement a subset of total requirements in each. Perform design through test/fix/delivery per build, successively adding capability.	Yes	Yes	Maybe
Evolutionary — Determine a set of user needs; final determination is unknown. Perform requirements definition through delivery. Refine user needs and system requirements and repeat for successive builds.	No	Yes	Yes
Other — Variations of the above or alternate approaches.			

e. Theater and tactical systems operate in a defined area of the operational theater and focus on combat and wartime missions. Test designs should address additional considerations related to operating in adverse operating environments, real-time processing constraints and tactical communications.

3–6. The software development strategy

The software development strategy is an element of the system acquisition strategy. Computer resource items may be newly developed or reused, in whole or in part, from other development sources. The type and degree of testing needed to verify correct, reliable operation can vary for each type.

a. Newly developed software. Software built for the first time needs verification and testing that all functional and performance and design requirements allocated to it behave as specified and expected, including the interfaces to other software, hardware and human operators. The majority of procedures and activities described in this pamphlet pertain to newly developed software.

b. Nondevelopment items. The T&E community needs to carefully review and understand plans which incorporate NDI in the acquisition strategy. Documentation, evidence of testing, extent of software modifications and eventual software supportability are areas which impact T&E.

(1) The NDI approach uses products that have already been developed for another purpose or another user. An NDI component is chosen because it should be capable of meeting a set of system requirements. The NDI systems can be the means to field a system faster and cheaper with little or no new development effort. For many systems, computer hardware is NDI.

(2) Software frequently fits in the NDI category because it may be off-the-shelf. However, software in NDI systems may not always be entirely off-the-shelf but is often a combination of off-the-shelf and newly developed software. NDI software which requires modification or system integration must undergo software T&E.

(3) NDI software may or may not require Government developmental testing. With rare exceptions, operational testing must be conducted to ensure that the NDI software can support the overall mission.

(4) NDI executive software is validated during operational testing using a representative functional application. Normally, executive software cannot be released or used with an application until it has successfully completed an operational test. (Executive software includes compilers, utilities, operating systems, special customized system software, and so forth.)

(5) Unique NDI T&E considerations are—

(a) Developer's enhancements or adaptations to NDI software should perform correctly and not introduce new defects.

(b) Developer provided documentation should be adequate to permit suitable operation and maintenance of NDI software in its intended user and maintenance environment.

(c) In order to use previous test results of NDI in lieu of Government (re)testing, the conditions under which the NDI was tested must be sufficiently similar to the conditions under which the NDI will be used by the Army.

(d) When integrating multiple NDI components, T&E must demonstrate that system performance characteristics are met.

c. Reusable software. Software which is developed for the purpose of being reused in other applications with little or no further modification can be more costly and time consuming to produce and verify than components built without reuse in mind. Determining and documenting the scope and potential interfaces of reusable items, such as requirements, design or code, is a more rigorous and formal process. Consequently, so is the test strategy to verify the items the first time. However, incorporating well engineered reusable components in succeeding applications should reduce overall integration and test effort and time.

3–7. Prototypes

a. Prototypes are working models suitable for evaluating system design, performance, or production potential. Software prototypes are normally development tools, not a testing technique or a substitute for system development, testing, and configuration management. A prototype often concentrates on a specific subset of the total user requirements.

b. Informal releases of prototypes are encouraged for demonstrations to users and early looks at the user interface designs and to assist in refining requirements earlier in the development process. Using prototypes can also reveal difficulties or constraints in implementation that normally would not become evident until later in the process. Deficiencies can be handled with less impact to the total program when detected early.

c. In the event that long-term operational use of prototype software is planned, the prototype material must be properly tested, documented and accepted into the approved software baseline in the same manner as other software comprising the system.

3–8. Security certification

a. The software test program must accommodate the requirements of AR 380-19 regarding information security.

b. Examining the control of the procedures used during design and test to develop software is an integral part of the software certification and system accreditation process. AR 380-19 states—

(1) Software must be completely tested before becoming operational.

(2) Both valid and invalid data must be used for testing.

(3) Testing is not complete until all security mechanisms have been examined and expected results attained.

(4) Upon completion of maintenance or modification of software, independent testing and verification of the changes is required before returning the software to operation.

3–9. Metrics and T&E

a. Metrics are technical and management tools that can highlight potential problems or deficiencies in the software development process or its products. They provide quantitative and qualitative measures which help focus management attention and, if appropriate, resources on the prevention or correction of problems.

b. Metrics are an integral aspect of controlling and reporting software T&E activities and are required for Army software developments. Metrics measure and provide feedback, affecting both the product and process, enabling managers to continuously improve the process.

c. Metrics may be developed, collected, and used by many organizations (developers, evaluators, LCSEC, software engineers, testers, SQA, IV&V, and so forth). Metrics are reported to evaluators, PMs, PEOs, and review council decision authorities.

d. Chapter 10 details the characteristics and use of each of the metrics referenced throughout this pamphlet. Many of the metrics provide insight into the software's readiness for test and the progress of testing that has occurred.

3–10. Disciplines essential to effective T&E

Several key disciplines are essential to ensuring the right software products and systems are built, verified, and fielded. They are configuration management (CM), quality assurance (QA) and a closed-loop corrective action system. In brief, CM ensures only authorized changes are made to baselined products under controlled circumstances. QA monitors and evaluates the development process and resulting products for adherence to approved development procedures and the work statement. The corrective action system assures that detected problems are properly recorded and resolved.

Section III
Software T&E Methods

3–11. Tools and related techniques

a. This section briefly describes a variety of testing methods which can be used to detect errors, to develop sets of test data, and to monitor computer system resources. The list is not all inclusive but representative. The overall software T&E program should incorporate a number of complementary techniques. Automated assistance is available to support many of these methods, such as computer aided software engineering (CASE) tools. The generic term 'program' is used to represent the software entity under test. A method may be applicable to one or more types of software entities, such as a unit or CSC, or a physical subset of an entity, such as a procedure or file.

b. Figure 3–1, which follows the last method described, identifies software system development activities in which each T&E method is typically employed. A method may be useful in other activities as well, based on availability of automated support, available resources, or other relevant factors.

3–12. Static analysis techniques

Static analysis involves examining or analyzing a software product rather than executing code in order to find errors.

a. Reviews, walk-through and code inspections. These techniques apply the principle of visual inspection of portions of technical documentation to detect errors. The procedure typically involves a small working group of technical personnel who use requirements documents, specifications, program listings and standards as the basis for performing line-by-line code reading, doing walk-throughs of test inputs, tracing requirements from document to document or performing attribute checklist inspections. The clean room software engineering approach, for example, makes heavy use of static verification techniques and formal specification methods, to the extent that unit testing is no longer necessary and code is first tested at a system level.

b. Code auditors. A code auditor is a software program that examines the source code of other programs to determine whether prescribed programming standards and practices have been followed.

c. Interface checking. The flow of information and control within a system are areas where mistakes can occur, for example, by calling the wrong procedure or passing the incorrect data. Interface checkers are automated tools which can analyze a requirements specification, design specification, or code to detect errors in information or control passed between software components and modules.

d. Physical units checking. Physical unit checking involves using an automated tool to specify and check measurement units in computations. For example, computations involving different units which are not meaningful would be detected, such as adding feet and seconds. Strongly typed programming languages, such as Ada, provide this type of check when programs are compiled.

e. Data flow analysis. This automated technique detects whether or not sequential series of events occur in software execution.

f. Structure analyzers. Automated structure analyzers detect violations of control flow standards, such as improper calls to routines, infinite loops and incidents of recursion in source code or design language statements. Analyses may be reported in tabular or graphical form.

g. Cross reference programs. Cross-reference programs produce lists of data names and statement labels showing all places they are used in a program. Compilers may include cross reference generation options. These programs are useful for programming languages without structured programming syntax, such as early versions of FORTRAN or BASIC.

h. Input space partitioning techniques. Partitioning techniques emphasize the use of path analysis, domains, or partitions to build sets of test data. Path analysis generates a set of test data which will cause a selected path in software to be executed. Domain testing detects errors in software control flow which occur when an input follows the wrong path. Partition analysis detects missing path errors, incorrect operators, domain errors, and computation errors as well as generating a test data set that is sensitive to domain and computational errors.

i. Complexity analysis. This technique examines coded algorithms or programs to determine whether improvements in areas of correctness, number of operations required, amount of space used or code straightforwardness are possible. Chapter 10 describes several methods of complexity analysis.

3–13. Dynamic analysis techniques

a. Assertion testing. This technique requires the use of an assertion preprocessing tool, which generates executable assertion code embedded with the source code. Assertions are statements which specify the intent of input, output, intermediate steps of functions, and constraints.

b. Cause-effect graphing. This technique applies to test case design. It is used to systematically select a set of test cases (data) which have a high probability of detecting errors that exist in a program. The technique examines the inputs and combinations of inputs to a program and identifies the expected outputs. These inputs and outputs are derived through analysis of requirements specifications instead of code, providing an independent check of the code's implementation of the requirements.

c. Performance measurement techniques. These techniques include execution time and resource analysis. This involves monitoring software execution to locate code or throughput inefficiencies either by random sampling or by means of software probes. Monitored items may include number of central processing unit (CPU) cycles for groups of instructions, waiting times, control passing from one software component to another, memory paging times, and amount of memory or secondary storage space used.

d. Path and structural analysis. These tools monitor the number of times a specific portion of code is executed, the amounts of time involved, and other data. Portions of code are classified into three levels for the structural analysis: statements, branches and paths. Statement analysis is the least rigorous while path analysis is the most rigorous method. Generally, some form of structural analysis is instrumented as part of other dynamic analysis testing. Structural analysis can be effective for detecting computation, logic, data handling and output errors.

e. Interactive debugging techniques. Interactive testing aids are tools used to control and analyze a program while it is executing. The programmer can suspend program execution at any point to examine program status, view the values of selected variables and memory locations, modify the state of the executing program and trace the control flow of the executing program.

f. Random testing. This technique produces random samples of input data, executes the computer program using this data, then compares the generated output to the expected output. Knowledge of the actual input distribution may or may not be considered in the input generation process. One advantage of random testing is the exposure of the program to sequences and combinations of inputs which would not be expected to happen. These combinations frequently are among the first combinations of inputs experienced in the operational environment. This method cannot test all possible combinations of inputs and should not be the primary approach in the overall test strategy. It can be useful, however, in discovering unexpected program behavior.

g. Functional testing. This is the most commonly used testing approach. The objective is to test the software functions by executing the program with specific controlled input, thereby verifying the functions performed by that program. Errors which prevent the program from operating correctly can be detected through functional testing; however, use of this method alone does not guarantee a thorough test of the code nor an absence of errors.

h. Mutation analysis. This technique detects errors in a program and determines how well the program has been tested. It entails studying the behavior of a large collection of different versions of the same program, called derivatives, which have been systematically derived from the original program. The method involves introducing a small number of errors into a program at a time, and repeating this many times. If the data used to test the program and its mutant derivatives is complete, each mutant should produce a different output data set given the same input data set. If this occurs, the program under test is assumed to be correct within the limits of the assumption, and the test data set is assumed to be complete. Mutation analysis requires automated tools to produce realistic mutations and to examine the behavior of the programs under test. An expert human analyst should also scrutinize the derivatives. Large programs can require an enormous number of mutations to produce effective results. Mutation analysis is used at the point where a source program exists with a set of test data upon which the program is known to operate correctly.

i. Error seeding. Error seeding is a statistical approach to evaluating the comparative quantity of errors remaining in large software systems by estimating the effectiveness of testing. The technique is to introduce errors into the code, perform the test, then analyze the output to determine the percentage of the known errors which were detected. This percentage is then applied to overall error detection results to estimate the number of unknown errors which were not detected. Characteristics of seeded errors that escaped detection are analyzed and new test cases devised to check for those errors. Error seeding is particularly applicable to critical software where errors occurring in the end use environment would be intolerable.

j. Real-time testing. Real-time testing simulates the operational system environment. Generally, this technique is of value in producing stress and volume loads, creating test conditions which are difficult to produce in a controlled setting, and for situations where tests in the operational environment are not possible for reasons such as safety and prohibitive cost.

(1) *Modeling, simulation and stimulation.*

(a) Modeling typically refers to using mathematical representations to mimic behavior or characteristics of the external environment. In simulation, selected features of the behavior of one system (the target system) are represented by the behavior of another system (an abstract depiction of the target system) through the use of software. Stimulation requires using one or more devices to recreate the actions or stimuli needed to make the system under test react as if receiving actual data in its deployed environment. Models are often used in simulation and stimulation programs. Stimulation devices for AIS are typically called remote terminal emulators.

(b) Significant technical risk can be mitigated by early use of modeling and simulation in software T&E. This risk reduction depends highly on the validity of the model as compared to the actual system. When models or simulations will be used as a substantial basis for formal acquisition milestone decisions, they must first be accredited and validated by an activity independent of the model developer (see AR 5-11).

(c) Modeling and simulation can also be employed to explore the sensitivity of test results to conditions other than those experienced in testing. This use of modeling and simulation is not, however, a category of real-time testing as described above. See DA Pam 73-8 for this type of information.

(2) *Test beds.* A test bed refers to a specific environment established for the purpose of testing project software. It may range from a set of programs used with a static analysis test tool to a fully operational suite of target hardware surrounded by simulation and stimulation devices. Test bed is synonymous with the consolidated software standards' software test environment.

3–14. Symbolic testing

This method, also called symbolic execution, is applied to paths through programs. Input values are symbols that stand for sets of values rather than actual values. The symbolic execution is a substitution of symbolic values of variables in expressions for the variables themselves. This technique can be used to generate expressions which describe the cumulative effect of the computations which occur in a program path. Symbolic execution is used as a basis for data flow analysis and proof of correctness of computations. It is primarily useful for languages that are algebraic such as FORTRAN. Other languages, such as COBOL, are constructed in a manner which makes it difficult to generate symbolic input.

3–15. Formal analysis

This method involves a formal mathematical proof that given a specific input and programming language rules, the desired output will be obtained. It relies on mathematical assertions regarding the intent of the specification.

Activity	Reviews, Walk., Inspect.	Code Auditors	Interface Checking	Physical Units Checking	Data Flow Analysis	Structure Analyzers	Cross Reference Prog.	Input Space Partitioning	Complexity Analysis	Assertion Testing	Cause-Effect Graphing	Exec. Time & Resource	Path & Structural Anal.	Interactive Debugging	Random Testing	Functional Testing	Mutation Analysis	Error Seeding	Modeling, Sim., Stim.	Testbeds	Symbolic Testing	Formal Analysis
Planning & Oversight																						
S/W Devel. Environment																						
System Requirements Analysis																						X
System Design																						
S/W Requirements Analysis	X		X																			X
S/W Design	X		X	X		X																
S/W Implement. & Unit Testing	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X	X	
Unit Integration & Testing	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	
CSCI Qualification Testing												X							X			
CSCI/HWCI Integration & Testing												X						X	X	X		
System Qualification Testing																			X			
Developmental Testing																			X			
Operational Testing																			S	L		
S/W Fielding																						
S/W Transition																						
S/W Configuration Mgt																						
S/W Product Evaluation																						
S/W Quality Assurance	X	X																				
Corrective Action																						
Joint Reviews																						

T&E Method

X - T&E method is typically applied in this activity.
S - Technique may supplement, but not replace testing.
L - Applies to AIS Limited User Test only.

Figure 3-1. T&E methods and development activities

Chapter 4
Building the Software T&E Team

4–1. General
This chapter discusses the PM's T&E team. The acquisition planning performed by the PM requires input from software T&E organizations to ensure that a viable T&E program is established and maintained.

4–2. Objective
Early meetings with the PM, PEO, materiel developers (or contractors, if applicable), and the other members of the T&E team are paramount to achieving a comprehensive and integrated test program. These meetings address the operational and Government developmental test concepts. Issues discussed are: development strategy, configuration management, test bed design, facilities, instrumentation, test files, simulations, models, proposed test dates, test players, and the overall T&E concept.

4–3. Organizations and responsibilities
a. The specific organizations responsible for T&E activities are identified by the PM for each acquisition program. Many of the organizations must be involved during mission need determination in order to provide early planning, assessment, and structure to the system/software development and T&E.

b. Software T&E personnel are part of the PM's system acquisition team to provide technically knowledgeable support in the areas of software design, performance, and capabilities. Their purpose is to enhance and improve the exchange of information, prevent duplication of testing and data collection, and provide the PM with information to make decisions.

c. Table 4–1 identifies general categories of T&E team players, summarizes their responsibilities, and provides examples of specific Army organizations in each category. Membership and level of participation of team members varies based on a program's acquisition category and strategy, DOD oversight interest, and other factors.

4–4. Software T&E team members
a. These organizations are identified and assigned early so that software T&E planning is integrated into the project as part of the PM's total program and acquisition team. This enables software T&E personnel to determine the extent to which data can be shared and tests and evaluations can be combined. The software T&E team can better support the PM and the CE process by—

(1) Sharing evaluations.
(2) Identifying deficiencies early.
(3) Providing alternative T&E strategies.
(4) Keeping the PM and decision authorities informed.

b. Table 4–2 is an outline of software T&E team members and responsibilities. Figure 4–1 depicts a typical level of member involvement for a major program.

c. Software T&E teamwork is enhanced through the effective use of working groups as well as informal working relationships.

4–5. Independence in software T&E

Independence in testing and reporting channels promotes objectivity in T&E activities. There are three basic levels of independence.

a. Independence within the development organization includes QA, CM, and test personnel who report through a different chain of management than the software designers and coders.

b. Independence within the PM matrix organizations includes quality, IV&V, and test personnel who provide evaluation or testing for the PM, but report through a major subordinate command (MSC) or major Army command (MACOM) rather than the PM's chain of command.

c. Independent developmental and operational testers and evaluators. These personnel report findings to DA decision authorities.

4–6. Working groups

a. Function. In keeping with the integrated product and process development concept, a PM establishes and typically leads one or more working-level IPTs (WIPTs) to monitor various program activities and products. The WIPTs usually deal with a particular topic and bring together the necessary disciplines to address the topic as they are needed. Two example WIPTs to address the topics of testing and computer resources are described here. The organization presented is not mandatory, however the tasks the groups perform would need to be accomplished by some IPT for the program.

b. Test IPT. This group is comparable to the former TIWG. Detailed information regarding the composition, responsibilities, activities, and products of this group can be found in AR 70–1 and DA Pam 73–1 and DA Pam 73–2. These items are summarized below.

(1) The test IPT is the primary T&E team formed to manage and plan the total T&E effort. This is implemented primarily via the system's test and evaluation master plan (TEMP). The team structures the T&E program for the system and its software, and integrates the varying test, evaluation, and data requirements.

(2) The test IPT is usually established during mission need determination and concept exploration.

(3) The structure of the team and its principal members varies depending upon the type of system. Some systems may address software testing via a WIPT devoted to computer resources in general. A computer resources IPT would provide the software T&E support and input to the system test IPT.

(4) The principal members of the test IPT are the PM, independent testers and evaluators, user representatives, logisticians, post deployment support personnel, and trainers. When appropriate to the system, a survivability/lethality representative and threat integrator are also principal members. Software quality assurance representative, development tester, and system engineers may participate as associate members.

(5) Duties performed by the team include—

(a) Preparing the TEMP for the PM. This is carried out most effectively by assigning members responsibility for the parts of the TEMP. Parts I and II are prepared by the PM with inputs from the principal members. Part III is prepared by the developmental tester and the independent evaluator and may include input from SQA, SCM, IV&V agent, and the software developer. Part IV is prepared by the independent operational tester and the independent evaluator. Part V requires input from all members to determine T&E resources (for example, automated testing drivers, simulations, models used, users to be involved in testing, and so forth). The TEMP format is provided in DA Pam 73–2.

(b) Chartering other groups to provide specialized input and support as needed.

(c) Resolving routine problems, assisting in allocation of T&E resources, determining where data can be collected to answer issues and criteria, and discussing evaluation and test events.

(d) Assisting preparation of T&E portions of the acquisition strategy, requests for proposal (RFPs) and related contractual documents as well as assisting evaluation of developer proposals when there are T&E implications.

(e) Coordinating waivers of approved testing.

(f) Determining the level of evaluation needed when a system change—

– is not a response to a new or revised operational requirement, and

– is not a preplanned product improvement to fill an existing operational requirement, but

– the combat developer (CBTDEV) or functional proponent (FP) determines the change to have direct, or significant potential for, operational impact.

c. Computer resources IPT. A computer resources IPT, comparable to the former CRWG, plans, monitors, and implements aspects of the acquisition and maintenance of computer resources for the PM. This is described in detail in AR 70–1 and DA Pam 70–3.

(1) This team is usually established as soon as computer resources are determined to be part of the system.

(2) Membership of the computer resources IPT typically includes the PM, user representatives, post deployment support personnel (e.g., LCSEC), independent testers (developmental and operational), independent evaluators, software quality assurance (product assurance), and other representatives as required.

(3) Duties include, but are not limited to—

(a) Preparing and updating the system's CRLCMP. The CRLCMP describes the management strategy for software development, testing, and life-cycle support and is required in accordance with AR 70–1. CRLCMP content and format are outlined in DA Pam 70–3.

(b) Managing life cycle computer resources for the system.

(c) Monitoring and participating in software development and the T&E process.

(d) Supporting the test IPT (or equivalent). By means of a Memorandum of Agreement (MOA), a computer resources IPT identifies the products, analyses, and T&E support it will provide the test working group. This typically consists of—

– Providing input to the system TEMP.

– Providing software and computer resources expertise to the test IPT.

– Updating the test IPT periodically with the status of software T&E, the impacts of software deficiencies, the readiness of software to support further T&E events, software T&E metrics and other related factors.

– Reviewing all RFPs to ensure that software T&E factors are addressed. A checklist of software T&E factors to be addressed in this review is located in appendix B of this pamphlet.

– Serving on Source Selection Evaluation Boards (SSEBs). Members of these boards review bidders' responses to RFPs and assess the offeror's capability to develop quality software. Any member, with the exception of independent operational testers and operational evaluators, may participate in SSEBs.

Table 4–1
Responsibilities in T&E

Organization	Examples	T&E responsibility
Materiel Developer (MATDEV) Also known by other names including: Matrix Support, assigned Responsible Agency, Assigned System Developer, PM's Matrix Support, Developer, System's Engineers, SQA, SCM, IV&V, post deployment software support (PDSS)LSCEC/CDA, Software Engineers	Army Materiel Command major subordinate commands (Communications Electronics Command (CECOM), Missile Command (MICOM), Tank Automotive Command (TACOM), Aviation and Troop Command (ATCOM), Armament, Munitions Munitions and Chemical Command (AMCCOM)) Information Systems Command (Information Systems Engineering Command (ISEC)) Information Systems Support Command (ISSC), (Software Development Center-Washington (SDC-W), Software Development Center-Lee (SDC-L)) Corps of Engineers Medical Research and Development Command Space and Strategic Defense Command MACOMs (for assigned information systems only)	Research, development, T&E acquisition of assigned systems in response to approved user requirements. Primary T&E functions for ensuring support to the PM include— – T&E support in designing, planning, executing, assessing, and reporting technical T&E programs. – Effective and timely system integration during system development to allow for T&E of total system. – Provide adequate and efficient design reviews, audits, and quality assurance in support of the T&E program for the system being acquired. – Provide IV&V activities during software development.
User's Representative Also known as Combat Developer (CBTDEV) or Functional Proponent (FP) or Proponent Agency (PA)	Training and Doctrine Command (TRADOC) Corps of Engineers Director of Information Systems for Command, Control, Communications, and Computers (DISC4) Intelligence and Security Command (INSCOM) Medical Command Criminal Investigation Command Information Systems Command Any DA Staff section or MACOM may be an FP for AIS systems	Formulates doctrine, concepts, organization, materiel requirements and objectives, prioritize materiel needs, and represent the user in the materiel acquisition process. Coordinates with PM & MATDEV on matters pertaining to area of expertise. Staff agency responsible for the subject area in which IMA resources are used or planned for use, including automation in support of the function performed. Develops and documents Critical Operational Issues and Criteria.
Program Executive Officer (PEO)	PEO-C3S, PEO-STAMIS, PEO-IEW, PEO-FS, etc.	Responsible for administering a defined number of major or non-major acquisition programs. PEOs report to and receive direction from the Army Acquisition Executive (AAE).
PM Also known as Project Officer (non-major programs), Program/Project/Product Manager, Program Sponsor, System Manager, Operations Manager (during PDSS)	PM-CCTT, PM-OPTADS, PM-SADARM, PM-ABRAMS, PM-SIDPERS, PM-ILOGS, MACOMs, etc.	Chartered to conduct business on behalf of the Army. Reports to and receives direction from either PEO or AAE and is responsible for the centralized management of an acquisition program. Responsible for planning and executing comprehensive T&E program including TEMP preparation, coordination, distribution, maintenance; establishment of test IPT (or equivalent); conducting developmental test readiness review (DTRR), preparing developmental test readiness statement (DTRS) and operational test readiness statement (OTRS); assuring conduct of developmental T&E in accordance with AR 73-1; providing system support and training packages. Responsible (with matrix support) for continuous evaluation.
Developmental Tester	Test and Evaluation Command (TECOM) ISEC MACOMs who are MATDEVs For assigned AIS, Army Materiel Command (major subordinate command) may be different organization during PDSS (for example, SQA)	Army command or agency that plans and conducts Government developmental testing, including software testing, qualification testing, technical feasibility testing, and so forth. Tests are reported in accordance with AR 73-1.

Table 4–1
Responsibilities in T&E—Continued

Organization	Examples	T&E responsibility
Developmental Evaluator	ISEC	Command or agency that addresses acquisition of effective, supportable, and safe systems by assisting in engineering design and development, and determining the degree to which the technical characteristics of the system have been achieved. Performs continuous evaluation on assigned systems. Evaluations/assessments are made to PM and ASARC/MAISRC.
Operational Tester	Operational Test and Evaluation Command-Test and Experimentation Command (OPTEC-TEXCOM) INSCOM MACOMs for non-major AIS and others as assigned Information Systems Command (ISC) (ISEC) (only during PDSS of selected AIS systems)	Army command or agency that conducts EUTE, FDTE, IOT, FOT, LUT, UAT, supplemental site test (SST). All major systems require independent operational testers (independent of MATDEV, user, and PM).
System Evaluator	OPTEC-Operational Evaluation Command (OEC) INSCOM Medical Command (MEDCOM) (medical materiel only in accordance with AR 73-1 and AR 40-60)	Army command or agency that addresses effectiveness, suitability, and survivability of the acquired/developed systems by determining the degree to which the system's operational issues and criteria have been satisfied. Also addresses acquisition of effective, supportable, and safe systems by assisting in engineering design and development and by determining the degree to which the technical characteristics of the system have been achieved. Performs continuous evaluation of all assigned systems. Required to be independent of MATDEV, user, and PM. Directly reports evaluations to ASARC/MAISRC.

Table 4–2
Software T&E team members

Organization	Examples	Description/Responsibility Summary
Users' Representatives	See table 4-1	The designated combat developers, functional proponents and/or proponent agents. Define and refine user needs and requirements to the developer, T&E personnel, PM, and others. Integral to ensuring that the system meets the stated user needs and requirements and must be involved throughout the entire acquisition process as a constant monitor to achieve user needs.
Software Developer(s)	Central Design Activity (CDA) Center for Software Engineering (CSE) Life Cycle Software Engineering Centers (LCSEC) Contractors	Organization(s) designated to design and develop software. Identified when the concept evaluation requires advanced demonstration of software as part of the concept definition process and to reduce risk. Software developer(s) used during early phases may or may not be the same as those who will design and develop the production software. Responsible for testing the software during as stated in work directives or the contract.
Software Developer's internal QA	Army Quality Improvement Office (QIO)	Provide the internal QA functions for software design, development and some test activities.

Table 4–2
Software T&E team members—Continued

Organization	Examples	Description/Responsibility Summary
		Perform audits, evaluate metrics, and carry out comparable activities to those performed by PM's SQA personnel. Report to management other than the software coders/designers. They may all be part of the same contractor or developer, but do not report through the same chain.
Software quality assurance organization	ISEC	Generally matrix support to the PM. Responsible for providing software quality evaluations throughout the acquisition. As requested by PM, SQA may assist in conducting software reviews. Participate in software walk-throughs, audits, analysis of metrics, and other similar activities. Efforts also include management of or participation in the software test program and software support transition.
Software configuration management organization	ISEC	Generally matrix support to the PM. Responsible for performing software configuration management activities throughout the acquisition. As requested by PM, SCM may assist in conducting system configuration control boards (CCBs), participate in software CCBs and perform software configuration audits.
Post deployment support organization	CDAs LCSECs CSE Contractors	Army agency responsible for managing and/or performing software maintenance. Provides evaluation of software development risks throughout life cycle. Plans for software support, transition, and maintenance demonstrations and prepares software suitability statement for materiel release. May also assist the PM in managing software development effort, including presiding over software related reviews and managing or participating in the software test program and software support transition.
Independent verification and validation organizations	Government agency Contractor	Designated by the PM to provide additional or corroborative software design evaluations, witness developer tests, and perform other quality assessment activities based on PM's determination of software risk level. For complex and sophisticated systems, system evaluators use inputs provided by IV&V agencies to ensure that software is sufficiently mature. IV&V agent may not be a part of the development organization.
Independent test organizations	OPTEC-TEXCOM (operational) TECOM (developmental) ISEC	Provide test planning and coordination with software/system evaluators. Provide the PM with independent developmental testing or operational testing support throughout the life cycle. May not be a part of the development organization.
Independent evaluation/assessment organizations	OPTEC-OEC (operational) ISEC	Provide test planning and coordination with software/system testers. Provide the PM with evaluation support throughout the life cycle. For systems with no IV&V agent, evaluators may be required to prepare additional reports for system design reviews regarding software status relative to system objectives. Cannot be a part of the development organization.

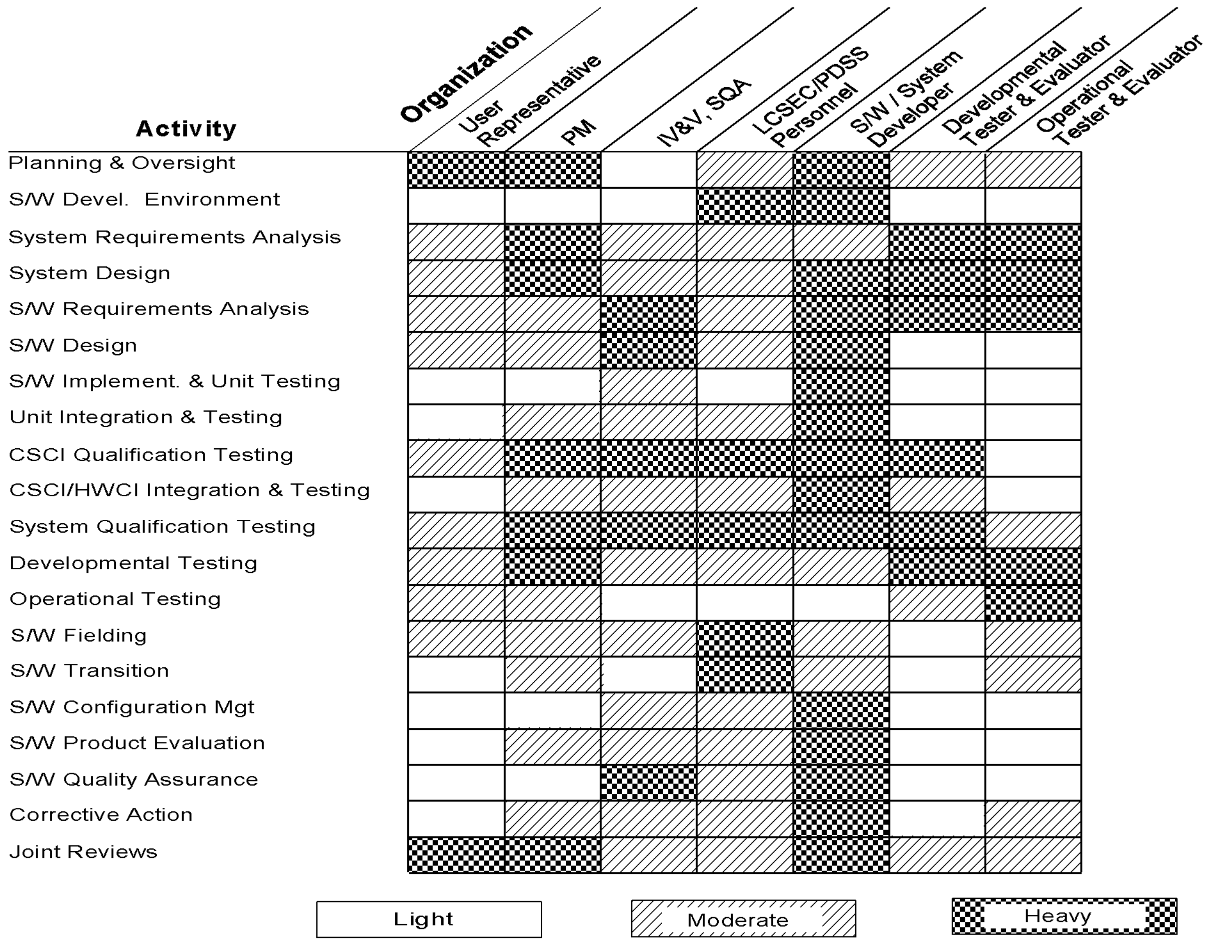


Figure 4-1. Level of T&E involvement

Chapter 5
Pretest Activities

Section I
General

5–1. Purpose
Well in advance of software development and testing, the need for a mission capability was identified and documented. Acquisition activity to respond to that need was initiated. This chapter summarizes these initial activities and identifies opportunities and procedures to enhance the software T&E process through CE.

5–2. Scope
a. The sections in this chapter parallel activities in the consolidated software standards of paragraph 2–2 *d*, but have been expanded to include T&E considerations beyond those of the developer. The events described here span mission need determination through the design of software entities.

b. The applicability of an activity to any given program and the depth to which it is carried out is dependent on overall system factors such as acquisition strategy and level of technical risk. These activities apply to both AIS and MSCR.

5–3. Objective
The objective of this chapter is to enhance identification of problems earlier in the life cycle in order correct them with less cost and disruption to the program. Continuous evaluation of pretest activities allows early assessment of the software development processes.

Section II
Planning and Oversight

5–4. General
a. Planning and oversight are integral and continuous processes that take place for the lifetime of a system.

b. Software system planning actions are shown separately from the core activities to which they pertain in order to emphasize their criticality regardless of deliverable contract data requirements list (CDRL) requirements. Completion of appropriate planning is a prerequisite to virtually every other activity in this pamphlet.

5–5. Objective
The purpose of planning and oversight is to prepare, record, and monitor the objectives, methods, and resources necessary to accomplish an orderly and effective system and software acquisition. The different members of the software T&E team perform this role in their respective areas of responsibility.

5–6. Entry criteria
For the purposes of this pamphlet, documented operational needs and requirements should exist to initiate this activity.

a. An operational need is identified by the users' representative, FP, or CBTDEV in the form of a mission needs statement (MNS).

b. An operational requirements document (ORD) describes the minimum essential: objectives, acceptable operational performance parameters, and critical technical characteristics needed to meet the MNS, for the most promising system concept. The ORD is prepared by the user or users' representative.

c. A UFD is required for AR 70–1 acquisition category I and II systems containing significant automated capabilities, and Class I-IV AR 25–3 systems for which TRADOC has CBTDEV responsibility. The UFD expands upon the automation aspects of the ORD and forms a basis for system requirements regarding computer resources. TRADOC Regulation 71–2 and TRADOC Pamphlet 71–7 provide more information in this area.

5–7. Test activities
a. Tests are not inherent during the planning and oversight activity, but may be arranged and performed at the discretion of the user, materiel developer (MATDEV) or FP, PM, and/or developer. These test activities may take the form of controlled experiments more often than formal tests.

b. Preparing and periodically updating plans is an important part of planning and oversight. Effective plans in many areas affect the ultimate acceptance, delivery and maintenance of an operational system and its support structure. To address the broader scope of CE, more than just test plans are summarized in this section. References for format, content or policy information are cited.

5–8. Test plans
a. Test and evaluation master plan (TEMP). Every Army acquisition program that falls under the purview of this pamphlet has a TEMP, with the exception of Class VI AR 25–3 information systems. The TEMP is a comprehensive system-wide test document that integrates all phases of a system's test program. The TEMP is a critical acquisition program management document against which technical progress and life-cycle decisions are assessed. Several significant elements of the plan are summarized below. It is essential that software-intensive materiel systems identify specific cumulative exit criteria for software, particularly in the critical technical parameters, for each testing phase in the TEMP to demonstrate continuing maturity growth prior to committing to major system tests. The PM is responsible for the TEMP and is assisted by the test IPT in its preparation (see para 4–6). Refer to AR 73–1 and DA Pam 73–2 for details (including software-specific TEMP considerations).

(1) Minimum acceptable operational performance requirements. These are the critical operational effectiveness, suitability, and survivability parameters and constraints that must be achieved by the system in order for it to be formally accepted by end users.

(2) Critical technical parameters. These are the measurable critical system characteristics that when achieved, allow the minimum acceptable operational performance requirements to be attained. These characteristics often depend on software.

(3) Critical operational issues. These are concerns related to operational effectiveness, suitability, and survivability that must be examined in operational test and evaluation to determine the system's capability to perform its mission. If every issue is resolved favorably, the system should be effective, suitable, and survivable when employed in its intended environment by typical users. Critical operational issues and criteria (COIC) are discussed in detail in DA Pam 73–3.

b. Software test plan (STP). The STP documents all the developer's plans for tests to qualify CSCIs and, if appropriate, qualify software systems. It includes descriptions of—

(1) Tests to be performed.

(2) Test environment.

(3) Test activity schedules.

(4) Data recording, reduction and analysis procedures.

(5) Traceability of tests to software requirements and system requirements .

c. Test design plan. The independent developmental evaluator is responsible for the TDP. This plan describes how the system's critical technical parameters will be tested during DT. Refer to DA Pam 73–4 for details.

d. Operational T&E plan. The operational tester and independent operational evaluator prepare the TEP. It describes how the system's operational issues will be tested during OT. The criteria against which the issues will be evaluated or assessed are included. Refer to DA Pam 73–5 for details.

5–9. Other plans
a. Computer resources life cycle management plan. This plan is prepared and maintained by the PM, often by means of a computer resources IPT (see para 4–6). A CRLCMP is required for software-intensive systems developed under AR 70–1 and serves as a memorandum of agreement between the MATDEV and responsible LCSEC. Comparable information is also required for AR 25–3 AIS and is documented in the system's Management Plan (MP) and System Decision Paper. In effect, the CRLCMP is the Government's software development plan for a system's computer resources over the life time of that system. It forms a basis for all strategy decisions involving the computer resources and integrates all aspects of computer resource acquisition and maintenance from resource identification to post deployment support. See AR 70–1, DA Pam 70–3 and AR 25–3 for details.

b. Software development plan. The SDP describes a developer's plans for performing a software development effort, whether that entails new development, reuse, re-engineering, maintenance or other software production activities. As directed by the acquirer, the SDP provides—

(1) The software development process to be used including a breakdown of planned builds, with the objectives and activities comprising each build.

(2) Overall software development methods and standards for each type of software product.

(3) Approaches for incorporating or developing reusable software products.

(4) How critical requirements will be handled.

(5) The approach employed to allocate and monitor use of computer hardware resources.

(6) Provisions for acquirer access to developer team facilities for reviewing software products and activities.

(7) Plans for conducting, monitoring and assessing risk for each detailed software development activity that applies to the effort.

(8) Methods and mechanisms to implement configuration management, quality assurance, internal product evaluation and corrective action on the processes and products described by the plan.

(9) A proposed set of joint management and joint technical reviews and approaches for their conduct.

(10) Plans for other related activities such as managing risk, managing subcontractors, handling security and privacy issues, interfacing with IV&V agents, applying software management indicators and metrics.

(11) A description of how the developer's organization supports the project and identifies resources.

(12) The organization of software activities into an integrated project network and all applicable schedules.

c. Software installation plan (SIP). The developer prepares a SIP when involved in installing software at user sites and the process is complex enough to require documented instructions. The tasks allocated to the developer, user, computer operations staff or others participating in the installation are described, including site specific information, if applicable. Tasks typically include preparing the site(s), documenting installation instructions, training users and converting from existing systems.

d. Software transition plan. The STrP is prepared by the developer and identifies the hardware, software and other resources needed for life-cycle support of deliverable software. It includes the developer's plans for transitioning deliverable items to the appropriate support agency and addresses training of support personnel.

e. Software QA plan. In this pamphlet, the software QA plan (SQAP) term refers to the documented policy, plans, and procedures of the acquirer, in this case the Government, for accomplishing the tasks comprising quality conformance assessment of software products and activities throughout a software-intensive system's lifetime. Software QA material may alternatively be documented in the overall system's QA plan. See chapter 8 for more information on quality assurance.

f. Software CM plan. In this pamphlet, the software CM plan (SCMP) term refers to the documented policy, plans, and procedures of the acquirer for accomplishing configuration management tasks involving software products and activities throughout a software-intensive system's lifetime. Software CM material may alternatively be documented in the overall system's CM plan. See chapter 8 for more information on configuration management.

g. Related plans. Other system level plans that may influence the implementation and test of software-intensive systems are—

(1) Basis of issue plan and qualitative and quantitative personnel requirements information. See AR 71–2, AR 71–9 and DA Pam 70–3 for the basis of issue plan (BOIP) and qualitative and quantitative personnel requirement information (QQPRI).

(2) Integrated logistics support plan. See AR 700–127, DA Pam 70–3 and DA Pam 700–55 for the integrated logistics support plan (ILSP).

(3) Reliability, availability, maintainability rationale report. See AR 71–9 and DA Pam 70–3 for the reliability, availability, maintainability (RAM) rationale report (RRR).

(4) System manpower and personnel integration management plan. See AR 602–2 and DA Pam 70–3 for the system manpower and personnel integration (MANPRINT) management plan (SMMP).

5–10. Evaluation activities

a. Per AR 73–1, the PM provides Army testers and evaluators the opportunity to participate in preparing the testing portion of requests for proposals to ensure that T&E requirements are adequately reflected in contractual documents. Likewise, software support personnel ensure that applicable computer resource policy is invoked in work statements and that deliverables are adequate for life time software maintenance.

b. The software T&E community provides input and review of in-house work directives and other work tasks relating to software development and T&E as well as contract statements of work (SOWs). A checklist for reviewing these documents and sample contract clauses are provided in appendix B. Examples of some of the issues assessed are—

(1) Collection or delivery of metrics.

(2) Ensuring requirements and design reviews are adequate and appropriate to the acquisition strategy.

(3) Delivery of software problem reports.

(4) Allowance for hooks, ports, etc. to enable collecting test data via instrumentation.

(5) Appropriate use of modeling or simulation in lieu of, or to augment, testing.

(6) Opportunities for user involvement and feedback during the development process.

c. The assessment of information in specific plans prepared by the developer is described in the Evaluation Activities paragraph of its corresponding activity elsewhere in this pamphlet.

d. Monitoring software products and processes against plans by members of the software T&E team occurs over the life of the system.

5–11. Metrics

The metrics marked with an x in table 5–1 apply to planning and oversight. All metrics used on a program should be routinely examined as part of the oversight process.

Table 5–1
Metrics applicable to planning and oversight

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
x	Complexity
x	Breadth of testing
x	Depth of testing
x	Fault profiles
x	Reliability

5–12. Decision criteria

Planning and oversight is an ongoing activity. However, the preparation, coordination and updating of plans identified above should be adequate and timely enough to support the activities guided by those plans. This condition is noted when a plan is identified as an entry criterion for an activity.

Section III
The Software Development Environment

5–13. General

a. During this activity, the organization that is to develop or perform maintenance on software, establishes a suitable environment to manufacture or sustain software products. The term" developer" used below is synonymous with" maintainer."

b. If a multiple build system or software acquisition strategy is in effect, this activity for a build means establishing the environment needed to complete that build.

5–14. Objective

The objective of this activity is to put in place the tools, controls and supplementary resources necessary to facilitate software production.

5–15. Entry criteria

Sufficient knowledge of the software production job to be performed and plans for the resources necessary to support it should occur prior to this activity. Plans for the software development environment were documented in the developer's SDP and the CRLCMP.

5–16. Test activities

Testing during this activity is typically done informally by the developer to confirm that elements of various development environments operate as intended.

5–17. Evaluation activities

a. CE actions consist of periodically assessing the plans and implementation status of applicable software development environments in the areas of software engineering, software test, software development library, software development files and nondeliverable software. The assessments are typically performed by PM, SQA, IV&V or LCSEC personnel.

b. Assessment criteria include determinations whether—

(1) Evidence exists that elements of an environment can perform their intended functions.

(2) Required elements of an environment are in place in advance of activities that will use them.

(3) Elements of, or information from, different environments are compatible or capable of being shared without excessive reformatting or re-entry of data.

(4) The environments are integral, not ancillary, to software production.

(5) Adequate and sufficient controls are in place to assure both development environment and software product integrity.

(6) The environments provide adequate information to manage software development.

5–18. Metrics

a. The metrics marked with an x in table 5–2 apply to the software development environment.

b. The schedule metric should show when critical items in the development environment will be available and usable and the activities that depend on them. While application dependent, examples of critical items are custom test fixtures and test articles, nondevelopment software, and training.

c. The software engineering environment (SEE) metric applies as an indicator of past ability to establish and maintain an adequate development environment.

d. The computer resource utilization (CRU) metric can be used to monitor utilization of development resources. Unanticipated demand for resources could result in delays in software production schedules or additional costs to augment the environment at a later time.

Table 5–2
Metrics applicable to software development environment

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
	Requirements traceability
	Requirements stability
	Design stability
	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles
	Reliability

5–19. Decision criteria

Representative products, documents and decision criteria typically addressed during the software development environment activity are shown in table 5–3.

Table 5–3
Software development environment decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer & PM	SDP	Adequate coverage of development environments and their support over the work period
S/W Developer & PM, Gov't. SQA or IV&V	Metrics Report(s)	Acceptable degrees of S/W development resource allocations and utilization

Section IV
System Requirements Analysis

5–20. General

a. The developer participates in defining and recording the requirements to be met by the system during this activity. This includes documenting the methods to be used to ensure that each requirement has been met.

b. If a multiple build system acquisition strategy is in effect, a system's requirements may not be fully defined until its final build. System requirements analysis for a given build, however, means defining the system requirements allocated to that build.

5–21. Objective

The objective of this activity is to identify and document the functional, performance, interface, and other acquirer-imposed requirements for a system.

5–22. Entry criteria

Defined and documented user requirements are a prerequisite to this activity.

5–23. Test activities

a. Tests are not inherent during system requirements analysis, but may be planned and performed at the discretion of the user, MATDEV or FP, PM, and/or developer. These test activities may take the form of controlled experiments more often than formal tests.

b. If system qualification testing applies to the build, the developer prepares a system test plan to document the system's qualification tests. For software-intensive systems and information systems, test requirements are typically documented in the STP.

5–24. Evaluation activities

a. Continuous evaluation activities performed within the T&E community need to include user representatives. Activities performed are—

(1) Review of the system's OCD to determine whether adequate analysis and understanding of user inputs, feedback and needs has taken place to ensure system requirements are accurate and complete.

(2) In-depth reviews of the SSS and interface requirements specification (IRS) for clear and complete descriptions of requirements regarding topics such as—

(a) Required states and modes of operation.

(b) Capabilities or functions of the system.

(c) External and internal interfaces.

(d) Installation specific dependencies.

(e) Safety, privacy and security.

(f) Use or incorporation of reused and nondevelopment items.

(g) Training and personnel related considerations.

(h) Priority, timing, sequencing, and criticality of requirements, functions, or interface characteristics.

(3) Assessment of requirements testability.

(4) Tracing from system requirements back to user requirements. The CBTDEV or FP participates in the requirements trace to ensure that the users' requirements have been properly interpreted by system specification writers.

(5) Maximum allowable use of computer resources and the conditions under which utilization measurements should be taken.

(6) Verifying that the ability to collect performance data during system-level tests, including formal Government tests, is addressed.

(6) Implementation and analysis of applicable metrics.

(7) Review of the draft STP for applicability and consistency of system's qualification test(s) with the system's SSS and IRS requirements.

b. If needed to resolve open issues or address areas of risk identified in the evaluation process, a formal system requirements review is appropriate.

5–25. Metrics

a. The metrics marked with an x in table 5–4 apply to system requirements analysis.

Table 5–4
Metrics applicable to system requirements analysis

Applies	Metric
	Cost
x	Schedule
	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
	Design stability
	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles
	Reliability

b. Requirements should show an acceptable degree of completeness, traceability and stability.

c. The SEE metric helps assess the capabilities of various software developers and how well their products meet requirements.

5–26. Decision criteria

Representative products, documents and decision criteria typically addressed during system requirements analysis are shown in table 5–5. Items marked " final" should contain comprehensive material that corresponds to the current build. Whether performed through a series of informal walk-throughs and assessments or by means of formal reviews, the issues of testability, performance, timing, and interfaces should be adequately addressed and documented prior to completing an iteration of the system requirements analysis activity.

Table 5–5
System requirements analysis decision criteria

Primary responsibility	Principal products affected	Decision criteria
User Representative and MATDEV	UFD	Final
PM, MATDEV and Developer	SSS, IRS [1] OCD	Draft Draft
Developer and PM	System requirements review(s), if required	Open issues resolved
System or S/W Developer & PM, Gov't. SCM, Gov't. SQA or IV & V	Requirements Trace(s) Metrics Reports	Updated Acceptable degrees of system requirements traceability and stability

Notes:
[1] System interface material may alternatively be documented in the IRS.

Section V
System Design

5–27. General

a. During this activity system requirements are iteratively defined and recorded by the developer as design decisions and implementation descriptions. This includes proof of traceability between system requirements and the hardware and software configuration items comprising the design.

b. The levels of system-wide design, system architectural design and detailed design may be documented in succession or in parallel depending on the quantity of units comprising the design and the complexity of their interactions.

c. If a multiple build system acquisition strategy is in effect, a system's design may not be fully defined until the final build. System design for a given build, however, means defining the design necessary to meet the requirements to be implemented in that build.

5–28. Objective

The objective of this activity is to define all design characteristics necessary to describe major system components as hardware configurations items (HWCIs), CSCIs or manual operations and the interactions among them.

5–29. Entry criteria

Sufficient system requirements analysis should occur prior to this activity in order to allocate system requirements to system design entities.

5–30. Test activities

Tests are not inherent during system design, but may be planned and performed at the discretion of the user, MATDEV or FP, PM, and/or developer. These test activities may take the form of controlled experiments more often than formal tests.

5–31. Evaluation activities

a. Continuous evaluation includes—

(1) In-depth reviews of system/subsystem design description (SSDD), interface design description (IDD), DBDD for clear and complete descriptions of system-wide decisions and a system architecture that, as applicable—

(a) Provides growth capability.

(b) Addresses interoperability requirements.

(c) Addresses safety, security, and privacy issues.

(d) Identifies component status such as: reused, reengineered, developed for reuse, new development, etc.

(e) Allocates computer hardware resources to design components and estimates capacity allotted to each component.

(f) Addresses all interfaces among system components as well as external system interfaces.

(2) Assessment of design testability.

(3) Tracing from system design components back to system requirements.

(4) Assuring user interface designs meet user requirements.

(5) Reviewing the use of nondeveloped software in the design to consider percent of modification, availability of documentation, and future testability or supportability issues.

(6) Implementation and analysis of applicable metrics.

b. If needed to resolve open issues or address areas of risk identified in the evaluation process, a formal system design review is appropriate.

5–32. Metrics

a. The metrics marked with an x in table 5–6 apply to system design.

Table 5–6
Metrics applicable to system design

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
	Design stability
	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles

Table 5–6
Metrics applicable to system design—Continued

Applies	Metric
	Reliability

b. Requirements should show an acceptable degree of completeness, traceability, and stability.

c. Specific computer resources are identified in this activity and allocations of their total capacities to the software design components are made.

d. The SEE metric helps assess the capabilities of various software developers and how well their products meet requirements.

5–33. Decision criteria

Representative products, documents, and decision criteria typically addressed during system design are shown in table 5–7. Items marked " final" should contain comprehensive material that corresponds to the current build. Whether performed through a series of informal walk-throughs and assessments or by means of formal reviews, the issues of testability, performance, interfaces and maintainability should be adequately addressed and documented prior to completing an iteration of the system design activity.

Table 5–7
System design decision criteria

Primary responsibility	Principal products affected	Decision criteria
Developer & PM	SSDD	Final
	IDD, DBDD [1]	Final (system design sections)
	System design review(s), if required	Open issues resolved
		When SSDD, IDD are approved, they become part of functional baseline
System or S/W Developer & PM, Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability; CRU allocations

Notes:
[1] IDD or DBDD material may alternatively be documented in the SSDD

Section VI
Software Requirements Analysis

5–34. General

a. During this activity the software requirements to be met by a CSCI are defined and recorded by the developer. This includes documenting the methods to be used to ensure that each requirement has been met and proof of traceability between CSCI requirements and system requirements.

b. If a multiple build software acquisition strategy is in effect, a CSCI's requirements may not be fully defined until the final build. Software requirements analysis for a given build, however, means defining the CSCI requirements allocated to that build.

5–35. Objective

The objective of this activity is to identify and document the functional, performance, interface, and other acquirer-imposed requirements for a CSCI.

5–36. Entry criteria

Sufficient system requirements analysis and system design should have been completed prior to this activity in order to identify and allocate system requirements to software.

5–37. Test activities

a. Tests are not inherent during software requirements analysis but may be planned and performed at the discretion of the developer and PM.

b. If software qualification testing applies to the build, the developer prepares a preliminary STP in parallel with requirements analysis to document the CSCI's qualification tests (see sec II of this chapter).

5–38. Evaluation activities

a. Continuous evaluation activities performed within the T&E community need to include user representatives. Activities performed are—

(1) In-depth reviews of SRS, IRS for clear and complete descriptions of requirements regarding topics such as—

(a) Required states and modes of operation.

(b) Capabilities or functions the CSCI implements.

(c) External and internal interfaces.

(d) Installation specific dependencies.

(e) Safety, privacy and security.

(f) Maximum allowable use of computer resources and conditions under which computer resource measurements apply.

(g) Use or incorporation of reused and nondevelopment item software.

(h) Communication network, training, and personnel related considerations.

(i) Priority, timing, sequencing, and criticality of requirements, functions, or interface characteristics.

(j) Qualification method(s) for each requirement.

(2) Assessment of requirements testability.

(3) Tracing from software requirements back to system requirements. The CBTDEV or FP participates in the requirements trace to ensure that the users' requirements have been properly interpreted by software specification writers.

(4) Implementation and analysis of applicable metrics.

(5) Review of the draft STP for applicability and consistency of CSCI qualification test(s) with the CSCI's SRS, IRS requirements, and proposed qualification methods.

b. If needed to resolve open issues or address areas of risk identified in the evaluation process, a formal software requirements review is appropriate.

5–39. Metrics

a. The metrics marked with an x in table 5–8 apply to software requirements analysis.

Table 5–8
Metrics applicable to software requirements analysis

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
	Design stability
	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles
	Reliability

b. More specific allocations of computer resource utilization become available through this activity. The allocations should be compliant with the contract and higher level specifications.

c. Requirements should show an acceptable degree of completeness, traceability and stability.

d. The SEE metric helps assess the capabilities of various software developers and how well their products meet requirements.

5–40. Decision criteria

Representative products, documents, and decision criteria typically addressed during software requirements analysis are shown in table 5–9. Items marked " final" should contain comprehensive material that corresponds to the current build. Whether performed through a series of informal walkthroughs and assessments or by means of formal reviews, the issues of testability, performance, timing, and interfaces should be adequately addressed and documented prior to completing an iteration of the software requirements analysis activity.

Table 5–9
Software requirements analysis decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer & PM	SRS, IRS [1]	Final
	STP	Draft
	Software requirements review(s), if required	Open issues resolved
		When SRS, IRS, STP are approved, they become part of allocated baseline
S/W Developer & PM, Gov't. SCM, Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability; CRU allocations

Notes:
[1] IRS material may alternatively be documented in the SRS.

Section VII
Software Design

5–41. General

a. During this activity the software requirements of a CSCI are iteratively defined and recorded by the developer as design decisions and implementation descriptions. This includes proof of traceability between CSCI requirements and the software units comprising the design.

b. The levels of CSCI-wide design, CSCI architectural design and detailed design may be documented in succession or in parallel depending on the quantity of units comprising the design and the complexity of their interactions.

c. If a multiple build software acquisition strategy is in effect, a CSCI's design may not be fully defined until the final build. Software design for a given build, however, means defining the design necessary to meet the CSCI requirements to be implemented in that build.

5–42. Objective

The objective of this activity is to define all design characteristics necessary for the production of actual software entities.

5–43. Entry criteria

The CSCI should have successfully completed those aspects of its software requirements analysis activity necessary for the level of software design being performed.

5–44. Test activities

a. Specific tests are not inherent during software design, but may be planned and performed at the discretion of the developer and PM. Demonstrations of evolving capability by means of prototypes, mockups of user interfaces or other mechanisms are recommended, however, to elicit user feedback and changes in the design prior to significant commitment of design to code.

b. If software qualification testing applies to the build, the developer completes the STP to document the CSCI's qualification tests (see sec II of this chapter).

5–45. Evaluation activities

a. Continuous evaluation activities performed at this time focus on analysis of the proposed software design to accurately reflect its software requirements and to ascertain its technical adequacy to achieve allocated system requirements. To accomplish this includes—

(1) In-depth reviews of the software design description (SDD), IDD and DBDD for validity and completeness of the technical design, such as—

(a) Design decisions and conventions regarding inputs and outputs to other systems, HWCIs, CSCIs, and users.

(b) Structure and interrelationships of the units comprising the CSCI.

(c) Flow of data and execution control.

(d) Recovery from malfunctions and handling of unexpected conditions or data.

(e) Computer resource allocations for items such as storage, memory and communications/network equipment.

(2) Review of user interfaces for simplicity, logical sequencing, consistency, and other human factors aspects.

(3) Tracing software design entities back to software requirements.

(4) Implementation and analysis of applicable metrics.

(5) If software qualification testing applies to the build, the final STP is reviewed for applicability and consistency of CSCI qualification test(s) with the CSCI's SRS and IRS requirements.

b. If needed to resolve open issues or address areas of risk identified in the evaluation process, a formal software design review is appropriate.

5–46. Metrics

a. The metrics marked with an x in table 5–10 apply to software design.

Table 5–10
Metrics applicable to software design

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
x	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles
	Reliability

b. CRU allocations to lower levels of software should not exceed overall requirements and should be compliant with the allocations in the specifications.

c. Requirements should show an acceptable degree of completeness, traceability, and stability.

d. Program design language (PDL) is often used during this activity. If complexity values for units exceed PDL thresholds, redesign may be necessary unless an adequate rationale is given.

e. The SEE metric helps assess the capabilities of various software developers and how well their products meet requirements.

5–47. Decision criteria

Representative products, documents, and decision criteria typically addressed during software design are shown in table 5–11. Items marked " final" should contain comprehensive material that corresponds to the current build. A series of informal walk-throughs and assessments is recommended to address the majority of noncritical design issues. However, prior to completing an iteration of the software design activity, the issues of testability, flow of control and data, error recovery, and interfaces, particularly user interfaces, should be adequately addressed and documented.

Table 5–11
Software design decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer & PM	SDD, IDD, DBDD [1]	Draft [2]
	STP	Final [2]
	Preliminary design review(s), if required	Open issues resolved
S/W Developer & PM, Gov't. SQA or IV&V	Requirements Trace(s)	Updated [2]
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability; CRU allocations [2]
S/W Developer & PM	SDD, IDD, DBDD [1]	Final [3]
	Detailed design review(s), if required	Open issues resolved
S/W Developer & PM, Gov't. SQA or IV&V	Requirements Trace(s)	Updated [3]
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability; CRU allocations; design stability; complexity, if PDL is used [3]

Notes:
[1] IDD and DBDD material may alternatively be documented in the SDD.
[2] Preliminary design.
[3] Detailed design.

Chapter 6
Test Activities

Section I
General

6–1. Purpose

This chapter summarizes activities comprising software production and testing. Opportunities and procedures to enhance the process through CE are identified.

6–2. Scope

a. The sections in this chapter parallel the activities of the consolidated software standards of paragraph 2–2 *d*, but have been expanded to include considerations beyond those of the developer. The events described here span implementation of software units through Government operational testing.

b. The applicability of an activity to any given program and the depth to which it is carried out is dependent on overall system factors such as acquisition strategy and level of technical risk.

c. Many of the T&E activities described below are repeated for each iteration of the design. Test activities are also often combined when test objectives are compatible, the integrity of the results is not jeopardized, and independence of evaluations will not be compromised.

d. During initial levels of software development testing, the developer has control over the T&E process. Software quality assurance and verification and validation (V&V) efforts are significant during each level of testing. Responsibility and control of testing shifts to Government agencies, the acquirer and user, as testing progresses.

6–3. Objective

The objectives of test activities are to verify that the item under test operates predictably and reliably and to detect malfunctions or omissions.

Section II
Software Implementation and Unit Testing

6–4. General

a. The developer transforms the software's design into computer programs and data structures during this activity. Each implemented data unit or program unit is tested to cover all aspects of its detailed design. Unit testing is the lowest level of test executed on software.

b. If a multiple build software acquisition strategy is in effect, this activity for a CSCI is not complete until that CSCI's final build. Software implementation and unit testing for a build includes those units, or parts of units, needed to meet the requirements to be implemented in that CSCI's build.

6–5. Objective

The objective of this activity is to produce software program and data entities. The purpose of unit testing is to validate requirements expressed in the detailed design descriptions and software requirements specifications. In addition, unit testing is performed to ensure that all source statements in a unit have been executed, each conditional branch has been taken, and that all boundary values (for example, minimum-maximum values) and edit criteria are tested.

6–6. Entry criteria

The detailed design of a unit should be completed prior to its implementation and test.

6–7. Test activities

a. The developer establishes test cases, test procedures and test data for each software unit and records this information in the appropriate SDFs.

b. If a set of benchmark test files (BMTF) exists, it should also be used as test data.

c. The developer conducts unit testing in accordance with the test cases, procedures and data in the SDFs.

d. Results of unit testing are recorded in the SDFs.

e. Test results are analyzed, software revised and retested, and the SDFs and other software products updated based on the test results.

f. The operating environment for unit testing is usually a local test bed system.

6–8. Evaluation activities

a. To gain insight into the software developer's progress, SDFs are reviewed. These evaluations are usually conducted by an independent organization whose management structure is separate from the software developer's management. A developer's quality assurance group or the Government's SQA, V&V or IV&V organizations are examples. The evaluations verify that—

(1) Software is developed in accordance with the detailed design and applicable development and coding standards identified in the SDP.

(2) Results and analysis of unit testing are recorded in the SDFs and revisions to software products, including unit test cases, procedures, and data, continue to track requirements.

b. Static analysis, data flow analysis and code walk-throughs are performed by the developer to assess software modularity, quality, and maintainability.

c. Completed and published system documentation and training packages are not normally available for unit level testing. However, if such documentation is available, it should also be reviewed.

d. Implementation and analysis of applicable metrics.

6–9. Metrics

a. The metrics marked with an x in table 6–1 apply to software implementation and unit testing.

Table 6–1
Metrics applicable to software implementation and unit testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
x	Complexity
x	Breadth of testing
x	Depth of testing
x	Fault profiles
	Reliability

b. Measured values for computer resource utilization become available at this time.

c. With the start of unit testing, data to support the breadth and depth of testing metrics can be collected and analyzed.

d. As coded units are placed under project CM control, values for fault profiles and design stability become available as a result of testing.

e. If complexity values for coded units exceed thresholds, redesign should occur unless an adequate rationale is given.

6–10. Decision criteria

Representative products, documents and decision criteria typically addressed during implementation and unit testing are shown in table 6–2. Only units which have been successfully tested are permitted to be integrated into components or programs for the next level of testing.

Table 6–2
Software implementation and unit testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer	SDFs	Adequate evidence of unit development and testing
S/W Developer and Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability, computer resource utilization, design stability, breadth and depth of testing, fault profiles

Section III
Unit Integration and Testing

6–11. General

a. During this activity, the developer integrates two or more software units and tests the composite software to ensure it works as intended. This process continues until all units in a CSCI have been integrated and tested.

b. If a multiple build software acquisition strategy is in effect, this activity for a CSCI is not complete until that CSCI's final build. Unit integration and testing for a build means integrating software developed in the current build with other software developed in that and previous builds and testing the results.

c. Historical equivalent activities are: CSC integration and testing—MSCR; SDT module/program testing—AIS.

6–12. Objective

The objective of this activity is to produce a CSCI which has completed developer internal CSCI testing covering all aspects of CSCI-wide design and CSCI architectural design. All integrated units should accept valid inputs and produce correct outputs.

6–13. Entry criteria

The lowest level units should successfully complete the implementation and unit test activity prior to their integration with other units.

6–14. Test activities

a. The developer establishes test cases, test procedures and test data for conducting unit integration and testing and records this information in the appropriate SDFs.

b. Benchmark test files are used as test data, if available.

c. The developer conducts unit integration testing in accordance with the test cases, procedures, and data in the SDFs.

d. Results of unit integration testing are recorded in the SDFs.

e. Test results are analyzed, software revised and retested, and the SDFs and other software products updated based on the test results.

f. The operating environment for unit integration and testing is usually a local test bed system.

g. With much of the software integrated, limits and bounds are tested and multiple paths executed to ensure that integration is proceeding in a robust manner.

h. Tests addressing run time efficiency and stressing the software at the limits of its specified requirements are also performed.

i. All discrepancies, malfunctions, and errors should be documented in problem/change reports in accordance with paragraph 2–2 *f*.

j. If CSCI qualification testing applies to the build, the developer can document the appropriate qualification test cases in the software test description (STD) as part of the CSCI qualification test activity.

6–15. Evaluation activities

a. Unit integration and testing is a developer internal activity. To gain insight into this process, software quality assurance, IV&V, and V&V personnel generally perform evaluations on-site and report to the other members of the software T&E community. The SDFs are reviewed to verify—

(1) Software is integrated in accordance with the documented unit integration and test approach and procedures in the SDP.

(2) Results and analysis of unit integration and testing are recorded in the SDFs.

(3) The developer revises software products, including unit integration test cases, procedures, and data, based on test results. The developer also ensures that these revisions continue to address requirements.

b. The developer performs static analysis, data flow analysis and code walk-throughs to assess software modularity, quality, and maintainability.

c. Completed and published system documentation and training packages are not normally available for unit integration testing. However, if such documentation is available, it should also be reviewed.

d. Implementation and analysis of applicable metrics.

6–16. Metrics

a. The metrics marked with an x in table 6–3 apply to unit integration and testing.

b. Breadth and depth of testing become more refined in this activity.

c. As unit integration and testing proceeds, the design progress component of the design stability metric should indicate more and more units are being entered into CM.

6–17. Decision criteria

Representative products, documents and decision criteria typically addressed during unit integration and testing are shown in table 6–4.

Table 6–3
Metrics applicable to unit integration and testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
x	Complexity
x	Breadth of testing
x	Fault profiles
	Reliability

Table 6–4
Unit integration and testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer	SDFs	Adequate evidence of unit integration and testing
S/W Developer and Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability, computer resource utilization, design stability, breadth and depth of testing, fault profiles

Section IV
CSCI Qualification Testing

6–18. General

a. During this activity, the developer prepares and demonstrates all the test cases necessary to ensure compliance with the CSCI's software and interface requirements.

b. If a multiple build software acquisition strategy is in effect, this activity for a CSCI is not complete until that CSCI's final build, or possibly later builds involving items with which the CSCI is required to interface.

c. Historical equivalent activities are: CSCI Formal Qualification Test (FQT) - MSCR; SDT cycle/system testing (partial) - AIS.

6–19. Objective

The objective of CSCI qualification testing is to demonstrate to the acquirer the CSCI's ability to meet its requirements as specified in its software and interface requirements specifications.

6–20. Entry criteria

a. The CSCI should successfully complete unit integration and testing, including developer internal CSCI testing.

b. Test preparation effort, including STD preparation and dry run, should occur prior to running a formal test witnessed by the acquirer.

6–21. Test activities

a. The developer establishes test preparations, test cases, test procedures, and test data for CSCI qualification testing and records this information in the appropriate STD.

b. Benchmark test files are used as test data, if available.

c. Prior to an acquirer witnessed test, the developer should perform a dry run of the test in accordance with the test cases, procedures and data in the STD. The results are recorded in the appropriate SDFs and test cases or procedures are updated as needed.

d. The developer conducts CSCI qualification testing in accordance with the test cases, procedures, and data in the STD.

e. All discrepancies, malfunctions, and errors will be documented in problem/change reports in accordance with paragraph 2–2 *f,* and entered into the developer's corrective action system.

f. Results of CSCI qualification testing are recorded in a software test report (STR).

g. Test results are analyzed, software revised and retested at all necessary levels, and the SDFs and other software products updated based on the results. The acquirer should be notified in advance when qualification retesting is to occur.

h. The operating environment for CSCI qualification testing is usually a local test bed system. However, qualification on target or production representative system is preferred, particularly for embedded MSCR.

6–22. Evaluation activities

a. Continuous evaluation activities include—

(1) Review of the STD to ensure CSCI qualification test preparations, test cases, and test procedures are adequate to verify compliance with STP, SRSs and interface requirements specifications (IRSs).

(2) Assessment of test drivers for their ability to induce data and processing loads stated in the operational mode summary/mission profile (OMS/MP). See AR 71–9 for details on the OMS/MP.

(3) Ensuring traceability from each STD test case to its CSCI and software interface requirements and, conversely, from each CSCI and applicable software interface requirement to the test case(s) that address it.

b. Implementation and analysis of applicable metrics.

c. If needed to resolve open issues or address areas of risk identified in the evaluation process, a formal test readiness review is appropriate.

6–23. Metrics
The metrics marked with an x in table 6–5 apply to CSCI qualification testing.

Table 6–5
Metrics applicable to CSCI qualification testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
	Complexity
x	Breadth of testing
x	Depth of testing
x	Fault profiles
	Reliability

6–24. Decision criteria
Representative products, documents and decision criteria typically addressed during CSCI qualification testing are shown in table 6–6. Items marked " final" should contain comprehensive material that corresponds to the current build and level of qualification testing.

Table 6–6
CSCI qualification testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
PM & Developer with SQA and IV&V	Test readiness review(s), if required, to resolve open issues	Ready to perform CSCI qualification test(s)
S/W Developer	STD	Draft
		Dry run of CSCI qual. test in accordance with STD
	STD	Final
	STR	Final
S/W Developer and Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability, computer resource utilization, design stability, breadth and depth of testing, fault profiles

Section V
Integration and Testing of Computer Software Configuration Items and Hardware Configuration Items

6–25. General
a. The developer successively integrates two or more software or hardware configuration items and tests the composite groupings to ensure they work together as intended. This process continues until all configuration items in a system or subsystem have been integrated and tested. CSCI/HWCI integration and testing also applies to building a system from subsystems.

b. If a multiple build software or system acquisition strategy is in effect, this activity may not be complete until the final build. CSCI/HWCI integration and testing for a build means integrating software and hardware developed in the current build with other CSCI/HWCI developed in that build and previous builds, and testing the results.

c. Historical equivalent activities are: system integration testing - MSCR; SDT cycle/system testing (partial)—AIS.

6–26. Objective
The objective of this activity is to produce a system which has completed developer internal system testing and meets its architectural design requirements.

6–27. Entry criteria
a. The unit integration and testing for a CSCI should have successfully completed prior to the CSCI's integration with other configuration items.

b. If this activity is followed by system qualification testing, applicable CSCI qualification testing should also have occurred.

6–28. Test activities
a. The developer establishes test cases, test procedures and test data for conducting CSCI/HWCI integration and testing and records the information in the appropriate SDFs.

b. Benchmark test files are used as test data.

c. The developer conducts CSCI/HWCI integration testing in accordance with the test cases, procedures and data in the SDFs.

d. Results of CSCI/HWCI integration testing are recorded in the SDFs.

e. Test results are analyzed, software revised and retested, and the SDFs and other software products updated based on the test results.

f. The operating environment for CSCI/HWCI integration and testing usually consists of target or production representative hardware. This may be supplemented or substituted with local test bed hardware only with the acquirer's approval.

g. With many of the configuration items integrated, limits and bounds are tested, and multiple paths executed to ensure that integration is proceeding in a robust manner.

h. Tests addressing run time efficiency and stressing the software at the limits of its specified requirements are also performed.

i. All discrepancies, malfunctions, and errors should be documented in problem/change reports in accordance with paragraph 2–2 *f.*

j. If system qualification testing applies to the build, the developer can document the appropriate qualification test cases in the STD as part of the system qualification test activity.

6–29. Evaluation activities
a. CSCI/HWCI integration and testing is a developer internal activity. To gain insight into this process software quality assurance, IV&V, and V&V personnel generally perform evaluations on-site and report to the remainder of the software T&E community. SDFs are reviewed to verify—

(1) Software and hardware is integrated in accordance with the CSCI/HWCI integration and test approach and procedures identified in the SDP.

(2) Results and analysis of CSCI/HWCI integration and testing are recorded in the SDFs.

(3) The developer revises software products, including CSCI/HWCI integration test cases, procedures and data, based on test results. The developer also ensures that these revisions continue to address requirements.

b. The developer performs static analysis, data flow analysis and code walk-throughs to assess software modularity, quality, and maintainability.

c. Completed and published system documentation and training packages are not normally available for CSCI/HWCI integration testing. However, if such documentation is available, it should also be reviewed.

d. Implementation and analysis of applicable metrics.

6–30. Metrics

a. The metrics marked with an x in table 6–7 apply to CSCI/HWCI integration and testing.

Table 6–7
Metrics applicable to CSCI/HWCI integration and testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
	Complexity
x	Breadth of testing
	Depth of testing
x	Fault profiles
	Reliability

b. Breadth of testing becomes more refined in this activity.

c. The design progress component of the design stability metric should indicate many more units incorporated toward the final system or build configuration and that the design is becoming more stable.

6–31. Decision criteria

Representative products, documents and decision criteria typically addressed during CSCI/HWCI integration and testing are shown in table 6–8.

Table 6–8
CSCI/HWCI integration and testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer	SDFs	Adequate evidence of CSCI/HWCI integration and testing
S/W Developer and Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Updated
		Acceptable degrees of: requirements traceability and stability, computer resource utilization, design stability, breadth of testing, fault profiles

Section VI
System Qualification Testing

6–32. General

a. During this activity, the developer prepares for and demonstrates the test cases necessary to authenticate compliance with all applicable system requirements.

b. If a multiple build software or system acquisition strategy is in effect, this activity is not complete until the final build.

c. Historical equivalent activities are: System Integration Test (SIT)—MSCR; SDT cycle/system testing (partial)—AIS.

6–33. Objective

The objective of system qualification testing is to demonstrate to the acquirer the system's ability to meet its requirements as specified in its system and interface requirements specifications.

6–34. Entry criteria

a. The system should have successfully completed CSCI/HWCI integration and testing, including developer internal CSCI/HWCI testing.

b. Test preparation effort, including STD preparation and dry run, should occur prior to running a formal test witnessed by the acquirer.

6–35. Test activities

a. The developer establishes test preparations, test cases, test procedures, and test data for system qualification testing and records the information in the appropriate STD.

b. Benchmark test files are used as test data.

c. Prior to an acquirer witnessed test, the developer should perform a dry run of the test in accordance with the test cases, procedures and data in the STD. The results are recorded in the appropriate SDFs and test cases or procedures are updated if needed.

d. System qualification testing is conducted by the developer in accordance with the test cases, procedures, and data in the STD.

e. All discrepancies, malfunctions and errors will be documented in problem/change reports in accordance with paragraph 2–2 *f*, and entered into the developer's corrective action system.

f. Results of system qualification testing are recorded in an STR.

g. Test results are analyzed, software revised and retested at all necessary levels, and the SDFs and other software products are updated based on the results. The acquirer should be notified in advance when qualification retesting is to occur.

h. The operating environment for system qualification testing usually consists of target or production representative hardware. This may be supplemented or substituted with local test bed hardware only with the acquirer's approval.

6–36. Evaluation activities

a. Continuous evaluation activities include—

(1) Review of the STD to ensure system qualification test preparations, test cases, and test procedures are adequate to verify compliance with SSS and applicable IRS requirements.

(2) Assessment of test drivers for their ability to induce data and processing loads stated in the OMS/MP.

(3) Ensuring traceability from each STD test case to the system and interface requirements it addresses and conversely from each system and applicable interface requirement to the test case(s) that address it.

b. Implementation and analysis of applicable metrics.

c. If needed to resolve open issues or address areas of risk identified in the evaluation process, a formal test readiness review is appropriate.

6–37. Metrics

a. The metrics marked with an x in table 6–9 apply to system qualification testing.

Table 6–9
Metrics applicable to system qualification testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
	Complexity
x	Breadth of testing
	Depth of testing
x	Fault profiles
x	Reliability

b. Breadth of testing becomes more refined in this activity.

c. The design progress component of the design stability metric should indicate more units are being incorporated and that the design is becoming more stable over time.

d. Actual measurements for system reliability may become available during this activity.

6–38. Decision criteria

Representative products, documents, and decision criteria typically addressed during system qualification testing are shown in table 6–10. Items marked " final" should contain comprehensive material that corresponds to the current build and level of qualification testing.

Table 6–10
System qualification testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
PM & Developer with SQA and IV&V	Test readiness review(s), if required, to resolve open issues	Ready to perform system qualification test
S/W Developer	STD	Draft
		Dry run of system qual. test in accordance with STD
	STD	Final
	STR	Final
S/W Developer and Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability, CRU utilization, design stability, breadth of testing, fault profiles

Section VII
System Developmental Testing (DT)

6–39. General

a. The DT described here consists of a series of system level tests where the Government controls the T&E process. This section describes software T&E considerations for systems containing software.

b. The type and scope of a developmental test depends on the life-cycle phase in which it occurs, the test objectives or the milestone decision it supports. The procedures in this section should be tailored accordingly. Additional information regarding participants, responsibilities, activities, applicable documents and reviews in DT can be found in AR 73–1 and DA Pam 73–4. This section augments some software-specific aspects of, but does not replace, DA Pam 73–4.

c. If a multiple build software or system acquisition strategy is in effect, this activity is not complete until the software's or system's final build.

d. Specific developmental tests for materiel systems are identified in AR 73–1. The level of detail outlined in this section covers the comprehensive aspects of DT that supports a certification of readiness decision for dedicated operational test and evaluation. The AIS SQT is a system DT. In this pamphlet, an AIS SDT is considered a developer test since it is not managed and performed by Government testers. Refer to sections IV, V, and VI of this chapter in regard to the SDT.

6–40. Objective

The objective of DT is to demonstrate that the system is capable of meeting all its critical technical parameters as specified in Part I of the TEMP, to identify technological and design risks, and to determine readiness to proceed to system operational testing (if applicable). DT focuses on system requirements in order to verify system technical performance and the ability of the system to perform in the user environment. Critical to the success of this and future tests is the ability to drive or load the system software in accordance with the user's OMS/MP.

6–41. Entry criteria

The following must have occurred prior to beginning a formal developmental test—

a. Evidence of successful completion of the developer's software and system qualification tests.

b. An approved TEMP exists that has been updated to reflect the developmental test.

c. The software baseline for test has been identified with name and version identifiers and has been QA certified.

d. A safety assessment report (SAR) has been provided to the test organization. When developmental testing involves troops, a safety release must have been issued to the test organization by the appropriate release authority (see AR 73–1, AR 385–16 and DA Pam 73–4).

e. A DTRR has been held and the developmental test readiness statement indicates the testing may proceed.

f. Problems detected during previous testing which will have impact on a successful developmental test have been closed, or approval to waive or defer tests for those conditions has been received from the TEMP approval authority after coordination through the test IPT.

g. System documentation regarding software operation is in near final form. This includes computer operation and users manuals, conversion documentation, and training materials.

6–42. Test activities

a. For evaluated systems, a system evaluation plan (SEP) and event design plan (EDP) are prepared by the independent evaluator/tester.

b. The developmental tester prepares a DTP to execute the EDP. Figure 6–1 summarizes an approach for addressing software requirements in the context of DT. Figure 6–2 provides additional detail for test case development and subsequent evaluation. The DTP should address means of collecting and reporting data for system and software reliability, depth and breadth of testing, and fault profiles metrics.

c. The developmental tester coordinates all activities with the PM and provides guidance on the resources required to support testing.

d. The PM convenes a DTRR, prepares a DTRS, and submits it to the developmental tester. Figure 6–3 is a checklist of software related questions discussed at the DTRR.

Tailor the generic test areas below to the specific software application based on its system level requirements. The following approach addresses software requirements in the context of DT.

a. Trace critical system requirements to the software requirements.
b. Identify the input conditions possible for each software requirement to develop a test coverage matrix. The test coverage matrix has cells which represent input conditions and the tests addressing them.
c. Design and execute tests to address the most important test conditions. It will generally not be possible to provide coverage for all input conditions, so testing should focus on the more critical or more likely test conditions with respect to the expected deployment environment. Sample generic test areas, or issues, are:

Issue	Definition
Performance	How well the software supports system performance
Interoperability	The degree to which data is correctly exchanged and interpreted between systems
Usability	The effort required to learn the user interface with the software, to prepare input and to interpret output of the software
Maintainability	The effort required to modify the software
Safety	How well the software inhibits the system from engaging in unsafe action toward personnel, equipment or materiel
Security	How well the software safeguards classified information and handles unauthorized attempts at system/data access

Figure 6-1. Software/system generic DT issues

Issue	Evaluation Criteria
Performance	
System response time	Conformance to specified time tolerances
System accuracy	Correctness of system level decisions and the proximity of computations to expected results
Recovery/restart procedures	Users can overcome potential processing malfunctions
Conversion processes	Data handling procedures for LOB and ROB processing are described and executed in a correct manner
Robustness	Legal or illegal operator entries or procedures do not cause system degradation except as allowed IAW requirements
Repeatability	Consistent conditions or events produce consistent results
Interoperability	
Transmission verification	Acceptance of legal transmissions and rejection of illegal transmissions
Transmission prioritization	Transmissions sent or received are prioritized and handled in the proper order
Stress	Data and transaction volumes, loads, varying conditions, or peak processing do not degrade the system except as allowed IAW requirements
Understandability	How well the user is able to manage the system including interactive terminal interface, cycle/system set-up, and input/output control
Interface considerations	Ease of data handling through cycle processing, intersystem data transfer, transmission of data over communications links, and time sharing links are functioning properly
Usability	
System response to user interaction	Acceptance of legal entries and rejection of illegal entries
Output product quality	Terminal displays, hard copy reports, magnetic tape and direct access files, etc. are correct and disposition and handling instructions for these products are clear and adequate
Training	Adequacy of appropriate training manuals, classroom and/or on-the-job instruction, and problem reporting procedures
Maintainability	
Documentation quality	Adequate degree of completeness, correctness, consistency and understandability of S/W documentation to maintain code
Code quality	Code quality is measured by programming style (e.g., complexity, modularity, commenting), reserve memory capacity and S/W metrics
Computer resources	Memory, processor, storage and network capacity is adequate to allow for anticipated growth
Safety	
Robustness	Legal or illegal operator entries or procedures, or loss of software capability do not cause system to exhibit hazardous conditions to personnel or materiel
Vulnerability	Degraded operating modes or recovery sequences do not cause undue safety problems for personnel except as allowed IAW requirements
Security	
Accessibility	Legal or illegal operator entries or procedures do not allow unauthorized use, manipulation or compromise of system or its data
Accountability	Attempts at unauthorized use or manipulation are detected and reported IAW requirements

Figure 6-2. Sample software issues and evaluation criteria

1. **General**.
   a. Has government reviewed/approved all software test plans/procedures/previous test results?
   b. Are all functional requirements clearly identified?
   c. Is there confidence that software functions will execute properly (walk-throughs, use of standards for requirements and design specifications, resource allocation, use of applicable standards from the approved Army technical architecture (ATA))?
   d. Is there a clear understanding of what software functions will be tested by the developmental and operational testers?
   e. Is the computer resources life cycle management plan current?
   f. Have plans been formulated to deliver all software documentation prior to DT/OT?

2. **Safety**. Does the system or software have any safety limitations (operational limitations for test personnel) either inside or outside the required performance envelope? If so, what corrective action has been taken or is any planned? Has a safety release been approved?

3. **Reliability, Availability, Maintainability**. Have software relevant failure definition/scoring criteria been established? Has software been identified as a potential source of failure?

4. **Configuration Management**.
   a. Has a technical documentation package been established for the preliminary product baseline?
   b. How is the software configuration being controlled?
      (1) Has a configuration management plan been approved which includes provisions for government approval of engineering change proposals-software and waivers/ deviations?
      (2) Has a configuration control board been established?
      (3) Are changes under configuration control?

5. **Testing**. Compare the requirements document against test results to date. There must be reasonable assurance that the system can satisfactorily pass technical tests or equivalent independent government tests.
   a. Do test results show that system and software requirements will be met? (Show quantity tested, what tests were conducted, relevant metrics and results).
   b. Have the tests addressed all system and software requirements?
   c. What problems have occurred and how have they been resolved?
   d. Have critical/major test incident reports from developmental and operational testing been closed out? (List and summarize corrective action.)

6. **Integrated Logistics Support**.
   a. Supportability.
      (1) Has the PDSS agent been identified?
      (2) Has the software maintainability evaluation been conducted?
   b. Training. What version of software was used for training?

7. **Security**. Is certification testing of system security planned?

8. **Test Resources**. Are any unique facilities, equipment or software instrumentation required and will they be available at the test site(s)?

Figure 6-3. DTRR software T&E checklist

e. Test data consists of live data files (when they exist) supplemented with user prepared data. Test data should be representative of typical and peak load operational conditions. When required, additional files to test high risk performance parameters should be included.

f. DT is conducted by the developmental tester. An ad hoc group of system users or operators may also participate based on test design.

g. DT is performed on target hardware.

h. Problems discovered during DT are recorded as system/software anomalies using the TIR. The report has a developer's analysis section used for corrective action reporting. All software problems including problems with test procedures will be recorded in accordance with paragraph 2–2 *f.*

i. The tester prepares a developmental test report describing the results of test execution and data collection efforts. The report is submitted to the PM, independent evaluators and appropriate review body (ASARC, in-process review (IPR) or MAISRC).

j. The independent evaluator prepares a system evaluation report (SER) or system assessment (SA) and submits it to the PM and appropriate review body.

k. Following the test, results are analyzed, software revised if necessary and retested, and the SDFs and other software products updated based on the test results.

l. During the DT, the QA certified software baseline should not be modified for use in the DT without prior approval of the PM and independent evaluators.

6–43. Evaluation activities

a. Results of testing are prioritized and categorized by a Government data authentication group. Principal members are the PM, user representative, evaluator, and developmental tester.

b. Test results are evaluated by the evaluator in accordance with the SEP.

c. Evaluation is a comprehensive V&V process conducted to ensure that all capabilities and requirements of the system are exercised and analyzed in accordance with the issues and criteria stated in the SEP and TEMP. Elements of the evaluation may include but are not limited to the six generic test issues of figure 6–1: software performance, interoperability, usability, maintainability, safety, and security. Sample subissues and evaluation criteria for each issue are shown in figure 6–2.

d. The evaluation should include discussion of relevant software metrics and any outstanding software problems. For example, software faults discovered and closed, test adequacy, and failure patterns over the DT period should be addressed.

6–44. Metrics

a. The metrics marked with an x in table 6–11 apply to system developmental testing.

Table 6–11
Metrics applicable to system developmental testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
x	Complexity
x	Breadth of testing
x	Depth of testing
x	Fault profiles
x	Reliability

b. Data from any of the metrics may be used to assist in determining readiness for developmental test.

c. Breadth and depth of testing, reliability, and fault profiles collected during the developmental test are of particular interest.

d. The traceability and stability metrics, complexity and CRU contribute towards the maintainability evaluation.

e. An examination of fault profiles for the period prior to DT shows the ability of the developer to identify and correct software problems. This examination also provides an indication of software maintainability.

f. The SEE metric may also provide insight regarding the developer's ability to maintain the software.

6–45. Decision criteria

Representative products, documents, and decision criteria typically addressed during system developmental testing are shown in table 6–12.

Table 6–12
System developmental testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
PM	DTRR, DTRS	Ready to perform DT
PM and Gov't. SCM	Executable S/W	S/W baseline for DT
Developmental Tester	Test Report	Final
Evaluator	SA	Final
LCSEC/PDSS agent	Maintainability evaluation	Draft
Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability, computer resource utilization, design stability, breadth and depth of testing, fault profiles, reliability

6–46. Other considerations

a. Preferably, there should be no open priority 1 or 2 problem/change reports or TIRs from previous testing prior to initiating DT. Severe test limitations may result if testing occurs with open problem reports. It may cause great portions of DT to be repeated due to invalid data.

b. Ideally, the software baseline used in training for the formal test should not be changed prior to the start of test. This reduces the risk of changes to RAM instrumentation, changes to data collection and reduction procedures, or that test participant retraining is necessary shortly before starting formal test.

c. The software baseline is not modified during DT unless severe problems are encountered, in order to maintain consistency of data collected throughout the test.

d. If the number of priority 1, 2, or 3 problems detected during DT become excessive, impacting the test objectives, the developmental tester can suspend or terminate testing in accordance with the policy stated in AR 73–1.

e. If the software baseline is modified during the test period, regression testing is required to ensure detected problems were corrected and additional problems were not introduced into the software.

f. It is very important to test interoperability (both intrasystem and intersystem) using actual target systems. The Army Interoperability Network (AIN) and the Digital Integration Laboratory

(DIL) at Fort Monmouth, New Jersey, and Joint Integrated Test Center (JITC) at Fort Huachuca, Arizona, are available for this purpose.

g. Post-deployment software support (PDSS) personnel should conduct the software maintainability evaluation.

Section VIII
System Operational Testing

6–47. General

a. Operational tests are system level tests where the Government controls the T&E process. This section describes software test and evaluation considerations for systems containing software. OT differs from DT in that OT—

(1) Is conducted on systems in an operational environment that is as realistic as practical.

(2) Employs personnel with the same skills and training as those who will operate, maintain and support the system when it is deployed, such as typical troops or user organizations.

b. The type and scope of an operational test depends on the lifecycle phase in which it occurs, the test objectives or the milestone decision it supports. The procedures in this section should be tailored accordingly. Additional information regarding participants, responsibilities, activities, applicable documents and reviews in OT can be found in AR 73–1 and DA Pam 73–5. This section augments some software-specific aspects of, but does not replace, DA Pam 73–5.

c. If a multiple build software or system acquisition strategy is in effect, this activity is not complete until the software's or system's final build.

d. This chapter outlines the comprehensive aspects of an IOT. When an accelerated software development acquisition strategy is used, the full level of detail described in this section applies to the IOT of the representative sample for fielding certification, IOT.C. Not all aspects of this section apply equally to the incremental software block IOTs that precede or follow the IOT.C, as they are dependent on the functionality of each block. See DA Pam 73–5 for detail.

6–48. Objective

The objective of this activity is to demonstrate that the system is capable of meeting its operational issues and criteria as specified in Part IV of the TEMP. OT focuses on how the system supports the user's mission, and the capability of the user to employ the system, often termed the system's operational effectiveness and suitability.

6–49. Entry criteria

The following must have occurred prior to beginning a formal operational test:

a. Evidence of successful completion of DT.

b. An approved TEMP exists that has been updated to reflect the operational test.

c. The software baseline for test has been identified with name and version identifiers and has been QA certified.

d. A safety release has been issued to the test organization by the appropriate release authority.

e. An OT pilot test has occurred and any deficiencies found in the DTP have been corrected.

f. One or more operational test readiness reviews (OTRRs) has been held and operational test readiness statements (OTRSs) in each review member's area of responsibility indicate the testing may proceed.

g. Problems detected during previous testing which will have impact on a successful operational test have been closed, or approval to waive or defer tests for those conditions has been received from the operational tester, operational evaluator, and TEMP approval authority after coordination through the test IPT.

h. System documentation regarding software operation is in final form. This includes computer operation and users manuals, conversion documentation, and training materials.

6–50. Test activities

a. For evaluated systems, a SEP is prepared jointly by the operational tester and evaluator.

b. The operational tester prepares an outline test plan (OTP) identifying all resources necessary to carry out the test.

c. The operational tester prepares a DTP to execute the SEP. Figure 6–4 summarizes an approach for addressing software requirements in the context of OT. Figure 6–2 provides additional detail for test case development and subsequent evaluation. The DTP should address providing TIRs to the PM to collect and report data for system and software reliability, breadth of testing, and fault profiles metrics.

d. The operational tester conducts an OTRR to identify any problems that may impact starting or adequately executing the operational test. The items outlined in figure 6–5 are appropriate OTRR discussion points regarding requirements software must meet to permit certification of readiness for dedicated operational testing.

e. A production database or equivalent is used as test data.

f. OT is conducted by the independent operational tester.

g. OT is performed on target hardware in the production representative system configuration. This includes requisite communications facilities, peripherals, and interfaces to other systems.

h. Problems discovered during OT are recorded as system/software anomalies using TIRs. All software problems, including problems with test procedures, will be recorded in accordance with paragraph 2–2 *e.*

i. The operational tester and evaluator prepare a system evaluation report (SER) assessing the results of the operational test. The independent evaluator evaluates the system's effectiveness, suitability, and survivability with respect to the critical operational issues and criteria. The report is submitted to the appropriate milestone decision review body.

j. Test results are analyzed, software revised if necessary and retested, and the SDFs and other software products updated based on the results.

k. Changes to the QA certified software or firmware baseline must not be implemented during the OT unless specifically acknowledged and concurrence received from the responsible operational test and evaluation agency. The software baseline is not modified during OT unless severe problems are encountered, in order to maintain consistency of data collected throughout the test. The commander of OPTEC, or MEDCOM in the case of medical materiel, must approve changes to the baseline.

6–51. Evaluation activities

a. The data from OT are reviewed and authenticated by a Government data authentication group. Principal members are the PM, user representative, independent evaluator, and operational tester.

b. Test results are evaluated by the independent evaluator in accordance with the SEP. If the system is found to be operationally effective and suitable, the T&E findings support a production decision or software fielding.

c. A completed software maintainability evaluation should be used as part of the system's suitability determination.

Tailor the generic test areas below to the specific software application based on system level requirements. The following approach addresses software requirements in the context of OT.

a. Trace critical operational issues to critical system requirements. Use the trace of critical system requirements to software requirements that was done during developmental test preparation to link the issues to software. Add any links from operational issues to software not covered by the critical system requirements.
b. Use the test coverage matrix and test results from DT to determine whether any cells in the matrix were not tested or incompletely tested to date. Add new cells to the DT coverage matrix for any new links to form an OT coverage matrix. Identify input conditions for each software requirement deemed critical.
c. Ensure that operational test designs address the most important test conditions. It will generally not be possible to provide coverage for all input conditions, so testing should focus on the more critical or more likely test conditions with respect to the expected deployment environment. Sample generic test areas, or issues, are:

Issue	Definition
Performance	How well the software supports system performance
Interoperability	The degree to which data is correctly exchanged and interpreted between systems
Usability	The effort required to learn the user interface with the software, to prepare input and to interpret output of the software
Maintainability	The effort required to modify the software
Safety	How well the software inhibits the system from engaging in unsafe action toward personnel, equipment or materiel
Security	How well the software safeguards classified information and handles unauthorized attempts at system/data access

Figure 6-4. Software/system generic OT issues

1. **T&E History**.
   a. Does the system possess any known priority 1 or 2 problems that impact the OT so as to constitute a deficiency relative to a critical operational issue?
   b. Have all priority 3 problems been documented, complete with appropriate impact analyses, relative to each problem's potential impact to the system's mission capability and ability to resolve the affected critical operational issues?
   c. Has the system functionality to be operationally tested and evaluated been made available prior to the start of OT?
   d. Has the system functionality to be operationally tested and evaluated been developmentally tested?
   e. Have features required to support system level requirements and the system interfaces required to interoperate with external systems been certified to be functional?
   f. Were the system features of item (e) certified in an operationally realistic environment against operational requirements?
   g. Are software requirements and design stable?
   h. Has sufficient depth and breadth of software and interface testing been performed?

2. **Safety**. Does the system or software have any safety limitations (operational limitations for test personnel) either inside or outside the required performance envelope? If so, what corrective action has been taken or planned? Has a security release been issued?

3. **Reliability, Availability, Maintainability**. Have failure definition/scoring criteria been established? Has software been identified as a potential source of failure?

4. **Configuration Management**.
   a. Is a deficiency identification, tracking and reporting system in place to support the monitoring of deficiency reports by the operational test agency?
   b. Has a software configuration management system with associated control procedures been put in place prior to the start of OT?
   c. Has the version of software to be used in the operational test been baselined?
   d. Will the operational test agency have complete access to the configuration management system during the operational test period?
   e. Will pending software or firmware changes, if any, be completed prior to the start of OT?
   f. Is a physical configuration audit of the software version to be fielded planned?

5. **Integrated Logistics Support**.
   a. Supportability.
      (1) Has the software maintainability evaluation been completed?
      (2) Was the maintainability evaluation performed by the PDSS agent?
      (3) Is the computer resources life cycle management plan current?
   b. Training. Will the materials used to train testers reflect the proposed operational test software baseline? If not, are workarounds in training needed? Have workarounds been approved?

6. **Security**. Has system security certification occurred? When is security accreditation planned?

7. **Test Resources**. Are any unique facilities, equipment or software instrumentation required and will they be available at the test site(s)?

Figure 6-5. OTRR software T&E checklist

d. Operational evaluation verifies and validates that typical user missions can be met by the system under test when operated in the deployed environment using typical personnel. System and software functions are exercised and analyzed in accordance with the COIC stated in the operational SEP and TEMP. Elements of the operational evaluation regarding software may include but are not limited to the six generic test issues of figure 6–4: software performance, interoperability, usability, maintainability, safety, and security.

e. The evaluation should include discussion of relevant software metrics and any other outstanding software problems. For example, software faults discovered, test adequacy, and reliability over the OT period should be addressed.

f. If the operational test is supporting a decision to release the software to users, the software fielding activity and its decision criteria should be used in the evaluation. See chapter 7 for the description of the software fielding activity.

6–52. Metrics

a. The metrics marked with an x in table 6–13 apply to system operational testing:

Table 6–13
Metrics applicable to operational testing

Applies	Metric
x	Cost
x	Schedule
x	Computer resource utilization
x	Software engineering environment
x	Requirements traceability
x	Requirements stability
x	Design stability
x	Complexity
x	Breadth of testing
x	Depth of testing
x	Fault profiles
x	Reliability

b. Data from any of the metrics may be used to assist in determining readiness for operational test.

c. Breadth of testing, reliability, and fault profiles collected during the operational test are of particular interest.

6–53. Decision criteria

Representative products, documents, and decision criteria typically addressed during system operational testing are shown in table 6–14.

6–54. Other considerations

a. In an accelerated development/fielding acquisition strategy, a limited user test is often conducted to prove out the operational test bed. This test bed is comprised of the target hardware and NDI software, such as commercial operating systems, database management systems, etc., that form the basis of the operational system. Application software is not usually tested at this time. As each application software block is developed, it undergoes an initial operational test on the test bed.

b. Integrated developmental and operational testing is encouraged if the independence of evaluations is retained and the integrity of results is not compromised. Integrated DT/OT is conducted simultaneously using the same hardware and software, occasionally with dedicated phases of DT and OT.

c. If OT is required to support PDSS, then a FOT is conducted by the operational tester. Otherwise, a user acceptance test is conducted by the FP or CBTDEV. For systems that have both a FP and a CBTDEV, the UAT is conducted by the FP. A UAT is limited in scope relative to an FOT with the primary purpose of verifying the functionality of changes to an information system in the user environment.

d. A supplemental site test (SST) may be necessary for information systems that execute in multiple hardware and operating system environments if there are differences between user locations that could affect performance or suitability. The SST supplements an IOT or UAT. The SST and UAT will not be used in lieu of an IOT to meet the requirements of DODD 5000.1 and AR 73–1.

Table 6–14
Operational testing decision criteria

Primary responsibility	Principal products affected	Decision criteria
PM and Gov't. SCM	Executable S/W	S/W baseline for OT
Operational Tester	OTRR, OTRS	Readiness to conduct OT
Operational Tester, Independent Evaluator	SA	Final
LCSEC/PDSS agent	Maintainability evaluation	Final
Gov't. SQA or IV&V	Requirements Trace(s)	Updated
	Metrics Report(s)	Acceptable degrees of: requirements traceability and stability; CRU utilization; design stability; breadth and depth of testing; fault profiles, reliability

e. There should be no open priority 1 or 2 problem/change reports or TIRs from previous testing prior to initiating OT. Severe test limitations may result if testing occurs with open problem reports. It may cause great portions of the OT to be repeated due to invalid data.

f. Ideally, the software baseline used in training for the formal test should not be changed prior to the start of test. This reduces the risk of changes to RAM instrumentation, changes to data collection and reduction procedures, or that test participant retraining is necessary shortly before starting formal test.

g. The software baseline is not modified during OT unless severe problems are encountered, in order to maintain consistency of data collected throughout the test.

h. If the software baseline is modified during the test period, regression testing is required to ensure detected problems were corrected and additional problems were not introduced into the software.

i. If the number of priority 1, 2, or 3 problems detected during OT become excessive, impacting the test objectives, the operational tester can suspend or terminate testing in accordance with the policy stated in AR 73–1.

Chapter 7
Activities Related to Fielding

Section I
General

7–1. Purpose

This chapter summarizes activities related to fielding software-intensive systems and transferring maintenance responsibility from the developer to a life cycle support agent. Opportunities and procedures to enhance the process through CE are identified.

7–2. Scope

a. The sections in this chapter parallel the activities of the consolidated software standards in paragraph 2–2 *d* but have been expanded to include considerations beyond those of the developer. The events described here span preparing the executable software and operating instructions for distribution through transferring the software product baseline to a maintenance agent for PDSS. The activities " preparing for software use" and " preparing for software transition" have been renamed " software fielding" and " software transition" to indicate their expanded scope.

b. The applicability of an activity to any given program and the depth to which it is carried out is dependent on overall system factors such as acquisition strategy and level of technical risk.

c. It is unlikely that many systems need all the manuals identified in this section. Substitution of commercial or other off-the-shelf documentation that adequately meets the intent of these manuals is encouraged.

d. As specified by the acquirer, this section also applies to training devices, automatic test equipment or other logistics support tools needed for the system. Installation and operating instructions as well as software maintenance information for these devices must also be passed to the life-cycle agent.

e. In embedded MSCR or logistics support equipment, information about installing and operating software is often incorporated into technical and field maintenance manuals. In that case, the software fielding activity includes developing or participating in documenting the appropriate sections of those manuals and associated training materials.

f. Elements of the DT, OT, software fielding, and software transition activities contribute to the materiel release decision.

7–3. Objective

The objective of activities related to fielding is to prepare and put in place the system's software logistics support environment. This includes replicating and disseminating software and operating instructions to users, training users and maintenance personnel, and establishing the environment necessary to repair and upgrade the software.

Section II
Software Fielding

7–4. General

a. The developer prepares the products discussed in this section, unless otherwise specified by the acquirer.

b. If a multiple build software acquisition strategy is in effect, planning should identify what software, if any, is to be fielded to users for each build. Software fielding for a build means those actions necessary to carry out the fielding plans for that build.

7–5. Objective

The objective of software fielding is to make the executable software available to users and deliver the manuals and instruction necessary to operate the software. Executable software includes any data files necessary to install and run the deployed software on target hardware, such as batch files and router tables.

7–6. Entry criteria

a. An approved software installation plan (SIP) or equivalent should exist to guide the installation process.

b. The software to be issued should show evidence of successful testing at all appropriate levels, must be accepted by the MATDEV/FP and user, and must have been certified by QA.

c. If materiel release provisions apply to the system, a request for release must be approved prior to actual field use. Paragraph 7–11, AR 700–142 and DA Pam 700–142 provide more detail on these requirements.

7–7. Test activities

Extensive testing is not inherent in preparing software packages for distribution. The products developed here are tested in other activities. Some check out is done during site installation.

7–8. Evaluation activities

Users and LCSEC/PDSS personnel should be heavily involved in continuous evaluation during this activity to—

a. Review the SIP to verify—

(1) Installation task descriptions identify the organization that will accomplish the task, such as user, developer, computer operations, PDSS personnel; the quantity of personnel, required skill levels, and installation schedule.

(2) Provisions for scheduling personnel that will comprise the installation team, students for training, computer support and technical assistance; and arrangements needed for facilities, lodging, and transportation, if required.

(3) Procedures are adequate and complete for—

(a) Installing the software.

(b) Checking out the installed software.

(c) Initializing databases or other software with site-specific data.

(d) Converting data from the current system.

(e) Performing a dry run of the procedures in operator and user manuals.

b. Review the software version description (SVD) to verify that the exact version of software prepared for each user site is identified. The SVD should provide—

(1) An inventory of materials comprising the version (tapes, disks, documentation, listings, etc.) along with applicable handling and security instructions or duplication and license restrictions.

(2) Explicit identification of all computer files making up the version.

(3) A list of all changes incorporated into the version since the previous version.

(4) Identification of any site unique data.

(5) Installation instructions and procedures for determining whether the version has been installed properly.

(6) Information on possible problems and known errors in the version. Instructions for recognizing, correcting or avoiding these problems should be included.

c. Review technical, maintenance or other operations manuals providing instructions for users who—

(1) Both operate and make use of the software's results, as in a software user manual (SUM).

(2) Prepare inputs to and receive outputs from the software but depend on others to operate the software in a computer center or other centralized or networked software installation, as in a software input/output manual (SIOM).

(3) Operate the software in a computer center or other centralized or networked software installation so that it can be used by others, as in a software center operator manual (SCOM).

(4) Operate the computers on which the software will run, as in a computer operation manual (COM).

d. Assess the manuals to determine their usability, correctness, and completeness in imparting the procedures necessary to—

(1) Set up the requisite hardware and software environment for use, including communications equipment.

(2) Operate and interpret results from diagnostic features.

(3) Perform mission tasks or computer runs in different operating modes, such as training, restart, emergency conditions, degraded modes, communications failures, manual override, shutdown or typical conditions.

(4) Identify, document and report problems or malfunctions.

(5) Recover from, work around or avoid malfunctions.

(6) Ensure continuity of operations.

e. Assure that suitable user training and support training is planned.

f. Ensure that installation occurs in accordance with the SIP.

g. Implementation and analysis of applicable metrics.

7–9. Metrics

The metrics marked with an x in table 7–1 apply to preparing for software use. Accounting for the cost of performing this activity and tracking a schedule of events, such as site installations and media preparation and distribution, are the only metrics associated with this activity.

Table 7–1
Metrics applicable to software fielding

Applies	Metric
x	Cost
x	Schedule
	Computer resource utilization
	Software engineering environment
	Requirements traceability
	Requirements stability
	Design stability
	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles
	Reliability

7–10. Decision criteria

Representative products, documents and decision criteria that typically should be met during preparation for software use are shown in table 7–2. Items marked " final" should contain comprehensive material that corresponds to the current build or release.

7–11. Other considerations

a. The materiel release process assures that Army materiel is suitable and supportable before the MATDEV may transfer accountability and control of the materiel to users. Systems containing software follow this process. Materiel release actions in support of new procurement, reprocurement, and system changes must also be supported by assessments or evaluations conducted by the independent developmental and operational evaluators. A software supportability statement is included in the materiel release package.

b. The following subparagraphs address software changes that fall under AR 70–142 materiel release provisions (whether embedded, proprietary or nondevelopment software). Adding, modifying or removing software is considered a change.

(1) Software that may significantly change the system's—

(a) Mission function.

(b) Mission capability.

(c) Performance parameters.

(d) Interoperability requirements.

(e) Software architecture.

(f) Maintainability.

(g) Reliability.

(h) Safety.

(2) A block update consisting of software changes of more than 30 percent source lines of code (SLOC), or 30 percent cumulative SLOC changes since the previous materiel release approval.

(3) A block update consisting of a software translation of 30 percent equivalent SLOC to a different computer programming language.

(4) Software that is significantly changed to run on a different computer processor or different computer system configuration.

(5) Software changes that require new test equipment for the user or impact 25 percent or more of the training program of instruction.

Table 7–2
Software fielding decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer	Executable S/W files [1]	Final
	SPS (exec. S/W section)	Draft
	SVD	Final (if applicable)
	SUM	Final (if applicable)
	SIOM	Final (if applicable)
	SCOM	Final (if applicable)
	COM	Final (if applicable)
	Applicable information for tech., maint. or training manuals	
S/W Developer and PM	Metrics Report(s)	Updates for cost and schedule
MATDEV, Materiel Release Review Board (MRRB)	Material Release	Approved by applicable decision authority

Notes:
[1] As identified in the executable software section of the SPS.

Section III
Software Transition

7–12. General

a. The developer prepares the products discussed in this section, unless otherwise noted.

b. If a multiple build software acquisition strategy is in effect, planning should identify what software, if any, is to be transitioned to the support agency for each build. Software transition for a build means those actions necessary to carry out the transition plans for that build.

7–13. Objective

a. This activity's objective is delivery of all end item executable software, associated source files, computer program support manuals and instruction necessary for the support agent to—

(1) Operate the deployed executable software on its target hardware.

(2) Regenerate the executable software.

b. Executable software includes any data files necessary to install and run the deployed software on target hardware, such as batch files and router tables. Source files, as used here, also include any ancillary data files essential to re-creating executable software from source materials.

7–14. Entry criteria

a. An approved STrP should exist to guide the developer's transition process.

b. An updated CRLCMP should exist to guide the support agent's transition process. Elements of the STrP may be incorporated into the CRLCMP by reference to reduce duplication.

c. Physical and functional configuration audits of software products to be delivered should occur prior to the completion of this activity for each build.

7–15. Test activities

Extensive testing of target software is not inherent in preparing software materials for transition. However, the developer should demonstrate to the acquirer that the deliverable software can be regenerated (for example, compiled, linked, loaded, into an executable product) and maintained using the hardware, software, and facilities identified in the STrP. Some check out is done as part of the support site installation process.

7–16. Evaluation activities

a. A software maintainability evaluation with subsequent supportability statement is required for materiel release. This is prepared by the LCSEC/PDSS agent.

b. LCSEC/PDSS personnel should be heavily involved in continuous evaluation during this activity to—

(1) Review the STrP to verify that all resources needed to control, copy, and distribute the software and its documentation, and to specify, design, implement, document, test, evaluate, control, copy, and distribute modifications to the software are identified and described. Resource descriptions include—

(a) Facilities (buildings, rooms, power, safety, security provisions).

(b) Hardware (models, versions, configurations, manuals, source of supply, licensing provisions).

(c) Software (names, version numbers, release numbers, configurations, manuals, vendor support, data rights).

(2) Ensure the STrP provides a schedule for transition activities, addresses training, and identifies number, type, skills levels, and security clearances required for support personnel.

(3) Assure that the SSDD reflects the " as built" system.

(4) Assure that the software product specification (SPS) is complete and up to date.

(5) Review the SVD to verify that the exact version of software prepared for the support site and each user site is identified. The SVD should provide—

(a) An inventory of materials comprising the version (tapes, disks, documentation, listings, etc.) along with applicable handling and security instructions or duplication and license restrictions.

(b) Explicit identification of all computer files making up the version.

(c) A list of all changes incorporated into the version since the previous version.

(d) Identification of any site unique data.

(e) Installation instructions and procedures for determining whether the version has been installed properly.

(f) Information on possible problems and known errors in the version. Instructions for recognizing, correcting or avoiding these problems should be included.

(6) Review software maintenance manuals providing instructions for support personnel who—

(a) Program the computers on which the software was developed or on which it will run, as in a computer programming manual.

(b) Program or reprogram firmware devices in which the software will be installed, as in a firmware support manual.

(7) As it applies to each support task, assess the manuals to determine their usability, correctness and completeness in imparting the procedures necessary to—

(a) Set up the requisite hardware and software programming environment.

(b) Operate and interpret results from diagnostic features.

(c) Describe the physical characteristics of the support equipment or target hardware, as applicable, that must be known to perform programming tasks. Examples are word lengths, interrupt capabilities, hardware operating modes, memory attributes, timers, clocks, input/output characteristics, sequencing requirements, and special features.

(d) Install, replace or repair firmware devices including contingencies to preserve continuity of operations when deployed.

(e) Ensure classification security is safeguarded.

(f) Identify, document, and report problems or malfunctions.

(g) Recover from, work around or avoid malfunctions.

(8) Assure that suitable support personnel training is planned, if applicable.

(9) Assure that a physical configuration audit occurs prior to acceptance of transitioning material identified in the SPS.

(10) Implementation and analysis of applicable metrics.

7–17. Metrics

The metrics marked with an x in table 7–3 apply to software transition. In addition to cost and schedule reporting, an assessment of software maintenance capability may be appropriate for organic or contracted support organizations whose comparable prior experience is limited.

Table 7–3
Metrics applicable to software transition

Applies	Metric
x	Cost
x	Schedule
	Computer resource utilization
x	Software engineering environment
	Requirements traceability
	Requirements stability
	Design stability
	Complexity
	Breadth of testing
	Depth of testing
	Fault profiles
	Reliability

7–18. Decision criteria

Representative products, documents, and decision criteria that typically should be met during preparation for software transition are shown in table 7–4. Items marked " final" should contain comprehensive material that corresponds to the current build.

Table 7–4
Software transition decision criteria

Primary responsibility	Principal products affected	Decision criteria
S/W Developer and Gov't. SCM	Executable S/W files	Final
	Source files	Final
	SPS	Final
	SVD	Final
	SSDD	Final (" as built" configuration
	CPM	Final (if applicable)
	FSM	Final (if applicable)
S/W Developer and PM, Gov't. SQA and Gov't. SCM	Functional configuration audit (FCA) and physical configuration audit (PCA)	Final
	Metrics Report(s)	Updates for cost and schedule; SEE if maint. capability unproven

Chapter 8
Ancillary Activities

8–1. Purpose
This chapter briefly describes activities integral to software development and maintenance that support the activities in chapters 5–7. The ancillary activities of this chapter provide many opportunities for obtaining CE information.

8–2. Scope
a. Activities in this chapter parallel the activities of the consolidated software standards of paragraph 2–2 *d,* but have been expanded to include considerations beyond those of the developer. The material covers major disciplines and events that facilitate other software activities to proceed effectively and efficiently.

b. The applicability of an activity to any given program and the depth to which it is carried out is dependent on overall system factors such as acquisition strategy and level of technical risk.

c. If a multiple build system or software acquisition strategy is in effect, the following activities should address the context and objectives appropriate for each build.

d. Each of the following activities need not be completely distinct from the others as long as appropriate independence criteria among them are maintained. For example, the quality assurance activity could perform all software product evaluations on a project.

e. Many of these activities are carried out at different levels by both acquirer (or acquirer's designee) and developer as tasks internal to their organization. Each organization operates under its own set of standard procedures and plans. For example, prior to delivery, the developer usually manages software source code files with internal CM procedures referenced in the SDP. When a product baseline is delivered and routine maintenance is transferred to a LCSEC/PDSS agent, the applicable procedures are typically documented in a CRLCMP or SCMP.

f. Virtually all the following activities are governed by the developer's SDP, or other plans incorporated to the SDP by reference, such as CM or QA plans. It is essential that the SDP be adequate to effectively guide those activities.

g. Other activities may be added to those identified in this chapter at the discretion of the PM and developer to address program specific areas of concern. Examples are risk management, subcontractor management, and interfacing with IV&V agents, other developers or working groups.

8–3. Objective
The objective of these ancillary activities is to ensure that the efforts and products of software development are controlled, consistent, and responsive to requirements.

8–4. Software configuration management
a. Effective CM is essential to support all other development activities. CM is composed of four basic elements: configuration identification, configuration control, configuration status accounting, and configuration audits.

b. Software configuration identification specifies all software related items to be placed under configuration control (for example, hardware, software, documentation, files, electronic media) and the scheme by which each item will be uniquely identified. The identification scheme must include the version, revision or release status of the item.

c. A baseline is a specific set of configuration items and their associated identifications that are under configuration control. Military Standard 973 defines the type of documentation that represent functional, allocated, and product baselines.

d. Configuration control is the systematic procedure by which changes to baselined CSCIs and HWCIs are proposed, justified, evaluated, coordinated, approved or disapproved, and implemented. This procedure is carried out by a configuration control board (CCB). CCB members, their level of authority to approve changes, and the steps to be followed to request authorization for changes, process change requests, track changes, distribute changes, and maintain past versions must be documented in the developer's SDP or CM plan. There may be several CCBs for a system such as the software developer's CCB, the Government subsystem CCB and Government system CCB. Distribution of changes may also include recovery of previously issued items, as required.

e. Configuration status accounting is the recording and reporting of information to manage configuration items which are under configuration control. This includes a record of the—

(1) Approved configuration documentation and identification markings.

(2) Status of proposed changes, deviation and waivers to the CSCI/HWCI (both engineering change proposals-software (ECP–Ss) and problem/change reports).

(3) The implementation status of approved changes.

(4) Changes made to an item since being placed under configuration control.

f. Configuration audits are formal examinations of one or more CSCIs/HWCIs to verify that the item under configuration control achieves the requirements allocated to it and that the technical documentation representing the item matches the physical implementation of the item. These are functional and physical configuration audits (FCA, PCA), respectively. Functional configuration audits and PCAs are typically conducted prior to acquirer acceptance of the product or " as-built" baseline.

g. DA Form 5005–R (Engineering Change Proposal—Software (ECP–S)), is recommended for requesting changes to baselined software. DA Form 5005–R is prescribed in AR 25–3. Guidance on using the form can be found in DA Pam 25–6.

h. The software developer is required to participate in all four elements of CM.

i. See MIL-STD–973 and DA Pam 25–6 for more detail on software CM.

j. The duration of this activity is the life time of the system.

k. The requirements stability and design stability metrics make use of information from configuration status accounting records.

8–5. Software product evaluation
a. Software product evaluations assess the quality of products built by the software development process. The developer is expected to perform on-going evaluations as products evolve and a final evaluation of each deliverable software product prior to its delivery.

b. Individuals evaluating a software product must be independent from the individuals that developed it.

c. The consolidated software standards of paragraph 2–2 *d* identify criteria and definitions for assessing a product's completeness, accuracy, understandability, consistency, compliance to contractual requirements, and additional factors, as appropriate for each product when performing the test and evaluation activities in chapters 5–7.

d. Problems detected are entered into the corrective action system.

e. Software product evaluation records are prepared and maintained to document the evaluations and their results.

f. The duration of this activity is the life time of the system.

g. Significant CE information can be gained from product evaluation records and from the T&E activities in chapters 5–7. This information should be made available to members of the appropriate WIPTs.

8–6. Software quality assurance
a. The software quality assurance activity monitors all software development activities and their products to ascertain compliance with procedural and acquirer imposed requirements. The latter are usually contractual requirements including standards invoked through the contract, while the former are prescribed in the developer's SDP or equivalent maintenance plan.

b. Individuals conducting QA evaluations must be independent from the individuals who developed the software product, performed the activity under review or are responsible for the software product or activity.

c. On-going audits, inspections or other QA evaluations are performed to assure that—

(1) Each software development activity that applies to the program, including the others in this chapter, is being performed in accordance with the SDP.

(2) Each software product undergoes software product evaluations, testing, and corrective action as required.

d. Problems detected are entered into the corrective action system.

e. Software QA records are prepared and maintained to document QA actions accomplished and their results.

f. The duration of this activity is the life time of the system.

8–7. Corrective action

a. The objective of the corrective action activity is to ensure a consistent and controlled process for tracking and resolving problems throughout the software development process. Problems and recommended changes to either software products or development activities should be handled by a corrective action system. Typically, items must be under project level or higher configuration control before the following requirements apply.

b. Requirements for a corrective action system are—

(1) Consistent use of standardized problem/change reports to document problems.

(2) Consistent classification and prioritization of software problems in accordance with paragraph 2–2 *f*.

(3) Analysis of data in the corrective action system to detect trends in reported problems.

(4) Periodic evaluations of corrective actions to determine whether problems have been resolved, adverse trends have been reversed and changes have been correctly implemented without introducing additional problems.

(5) The corrective action system be closed-loop. That is, all detected problems are promptly reported, entered into the system, action to resolve them is initiated, resolution is achieved, status of problems is regularly reported, and corrective action records are maintained.

c. The fault profiles, reliability, and requirements stability metrics examine various aspects of the corrective action system.

d. Different organizations may employ different corrective systems throughout the life of a project, however the requirements above apply to all corrective action systems.

8–8. Joint reviews

a. Joint technical or management reviews among developer and acquirer personnel are convened for the purpose of reviewing the status of the project or product, to surface and resolve outstanding issues, determine and concur on strategies to mitigate identified risks and to foster communication. It is recommended that users also participate, especially in demonstrations involving user interfaces, in order to elicit feedback based on the user's view from a total mission perspective. Specific reviews are not mandatory, but some degree of formal interaction between developer and acquirer is necessary and may be mutually agreed upon.

b. In addition to areas covered in item paragraph *a* above, technical reviews typically—

(1) Examine in process or final software products for accuracy, consistency, completeness, adequacy of testing information, and understandability.

(2) Review and demonstrate proposed technical solutions.

(3) Provide insight and obtain feedback on the technical effort.

(4) Surface and resolve technical issues.

(5) Identify near and long term risks with respect to quality, cost or schedule concerns. The software metrics discussed in chapter 10 of this pamphlet can contribute to identifying areas of technical risk.

c. In addition to areas covered in item paragraph *a* above, management reviews typically—

(1) Review overall project and software product status.

(2) Resolve open issues from technical reviews.

(3) Surface and resolve management issues.

(4) Obtain timely acquirer approvals and commitments needed to accomplish the project within schedule, cost or other constraints.

(5) Identify near and long term risks with respect to concerns not raised in technical reviews. The software metrics discussed in chapter 10 of this pamphlet can contribute to identifying areas of management risk.

d. Representative management reviews have been identified in the activity descriptions of chapters 5–7. Additional formal reviews that may be appropriate for some programs are: software plan review, operational concept reviews, test result reviews, software usability reviews, software supportability reviews, or critical requirement reviews. The consolidated software standards in paragraph 2–2 *d* identify candidate issues for resolution at these meetings.

e. Milestone decision reviews (MDRs) are required for all defense systems acquired via DODD 5000.1. Approval by the designated Government milestone decision authority is needed to commence the activities of the next life cycle phase. Figure 1–1 depicted the standard system decision milestones and phases. Milestones are sometimes tailored or combined as is most effective for a particular program. During a MDR, the status of all major aspects of the system's development progress are reviewed including DT and OT results to date and whether the TEMP's exit criteria from the current phase have been satisfactorily met. Movement to the next life cycle phase is recommended, as appropriate. Exit criteria for the next milestone review are also established. Decision authorities for major defense programs are the Defense Acquisition Board (DAB), ASARC, and MAISRC.

f. An in-process review may be requested by the milestone decision authority at any time. It is similar but narrower in scope than an MDR. The purpose of an IPR is to determine—

(1) Current program status.

(2) Progress since the last decision authority review.

(3) Program risk and risk reduction measures.

(4) Potential problems that require guidance.

g. The software metrics discussed in chapter 10 of this pamphlet can contribute to reporting program status at MDRs or IPRs.

8–9. Other considerations

Program, project, or product managers of designated major AIS (MAIS) programs must report quarterly through the PEO to the appropriate DOD chief information officer in accordance with DOD 5000.2–R. The status of the program, its progress, significant issues, risks and risk reducing strategies are accounted. Many of the CE activities and metrics discussed in this pamphlet can contribute directly to the PM's assessment of—

a. Schedule and progress.

b. Growth and stability.

c. Funding and personnel resources.

d. Product quality.

e. Software development performance.

f. Technical adequacy.

See chapter 10 for an illustration that correlates specific metrics to MAIS assessment issues.

Chapter 9
Post Deployment Software Support Considerations

9–1. Purpose

Post deployment software support refers to modifications or upgrades made to a system's software following the system's MS III decision review and initial fielding. This chapter outlines issues pertinent to PDSS and approaches for addressing those issues.

9–2. Scope

a. This chapter applies to the production and deployment, operations, and support life-cycle phases.

b. System modifications and upgrades include multi-system changes, block changes, preplanned product improvements, class I

ECPs, and system change packages. In this chapter, the modifications of software and computer resources, regardless of how the change is implemented, is referred to as a software change package.

c. System changes that are extensive enough to warrant approval as a major modification in a post MS III decision review are not considered PDSS, but a variation of a new program start. The milestone decision authority determines which acquisition phase the program should enter.

d. The applicability of procedures in this chapter to any given program and the extent to which they are carried out is dependent on overall system factors, such as deployment philosophy, the criticality and urgency of a change.

9–3. Objective

The objective of PDSS is to correct deficiencies. Deficiencies include both problems reported by users or detected during software maintenance, and modifications needed to improve system software to meet new or changed requirements.

9–4. PDSS issues

a. The PDSS environment generally produces many small changes over a period of time rather than a few large changes. The PDSS organization typically collects these changes into a few formal software releases to avoid disrupting the fielded system. Differences in the amount of change to software and timing of software releases should be considered in identifying the scope of total T&E required and the extent of T&E team involvement.

b. Software development activities performed in PDSS are the same as those carried out prior to first fielding. They are tailored as appropriate, however, to reflect the effort required to implement each SCP, update pertinent documentation, verify the SCP, and issue changes to users. The scope of the change and the criticality of affected software units should be considered in determining the SCP's T&E strategy.

c. If a SCP does not have operational impact, then the PDSS agent determines the action necessary to support the decision to field the change. The maintenance PM determines—

(1) The scope of software change in the SCP.

(2) The amount of rework necessary to implement the changes.

(3) The amount of testing needed to ensure that new or modified functions operate properly and that no new errors have been introduced.

d. Changes that introduce new or revised operational requirements or changes that may have an operational impact on the system require independent developmental and operational evaluations. Testing must provide the information needed to evaluate the impact of the change.

e. The urgency of delivering a change to user agencies may have an impact on the extent and thoroughness of a given T&E effort.

9–5. Controlling software changes

a. Changes to the software production baseline are documented on ECP–S and categorized based on the urgency of implementation in accordance with MIL-STD–973 (emergency, urgent or routine). They are also prioritized in accordance with paragraph 2–2 *f* relative to the impact on operational mission effectiveness.

b. An ECP–S often addresses a set of related problem/change reports. Packages of changes are approved and scheduled for implementation by the appropriate CCBs (see chap 8).

9–6. Scope of testing

a. The developer performs software unit testing and unit integration and testing of the new or modified software units.

b. The developer should repeat some or all aspects of qualification testing (CSCI and system) to demonstrate that previous requirements are unaffected and new or modified requirements are met.

c. When independent developmental or operational evaluations are necessary, the procedure outlined in paragraph 9–7 can assist in determining the level of DT/OT needed to support those evaluations. In general, these evaluations are needed when changes in computer resources (hardware, software, firmware or communications):

(1) Have a physical impact on either the operation or support of the system.

(2) Have a noticeable impact on the system's operational effectiveness, suitability, and survivability, affect user interfaces, or impact critical mission functions.

(3) Cumulatively affect 15% or more of the software units in the system since the last time such evaluations were made.

9–7. Determining test support needed for independent evaluation

a. The procedure described in this paragraph assesses various aspects of the deployed system's T&E history, current maintenance environment and potential impact of the SCP on the system's operational effectiveness and suitability. The intimate knowledge and informed judgment of the test IPT and CCB principals should guide the decisions made in applying the procedure described in this paragraph and in interpreting its results.

b. There are several steps in the procedure:

(1) Determine the potential problems for a SCP using figure 9–1.

(2) Determine the likelihood of each problem using table 9–1.

(3) Determine the severity of each problem using table 9–2.

(4) Combine the findings of tables 9–1 and 9–2 to determine the amount of testing needed for that problem using table 9–3.

(5) Tailor the DT and OT measures of performance (MOPs) and measures of effectiveness (MOEs) to address the problem.

c. Examine all DT and OT MOPs needed to adequately test the SCP to plan the necessary test events. It is the responsibility of the evaluator to determine the most effective mix of DT and OT to support their evaluations. This could entail substantial use of developer test information, concurrent DT/OT exercises, simulations, or other strategies. See chapters 3 and 6 of this pamphlet and DA Pam 73–4 and DA Pam 73–5 for guidance on planning developmental and operational tests.

d. It is recommended that the figure 9–1 checklists be used several times during the course of SCP planning and implementation to improve the estimate as more information becomes known. The last check should contain no " unknown" answers—mark these as " yes" to represent worst case.

Table 9–1
Determining problem likelihood

Probability of problem is	When the problem will—
Very High	Occur frequently in the system's life
High	Occur several times in the system's life
Medium	Likely occur at some time in the system's life
Low	Probably not occur in the system's life, but may occur

Table 9–2
Determining problem impact

Impact of problem is—	If the problem causes—
Catastrophic	Mission failure, loss of system or loss of personnel
Major	Severe mission degradation, personnel injury or system damage
Minor	Slight mission degradation, personnel injury or system damage
Negligible	Less than minor personnel injury or system damage, no mission degradation

9–8. Other considerations

a. System post deployment review.

(1) The PM should plan to convene one or more system post-deployment reviews (SPRs) during PDSS to determine how well the system is functioning. The first SPR is recommended approximately 6 months after all initial units are equipped or all site installation is completed. The review should assess—

(a) How well the operational system is satisfying user requirements to meet the stated mission.

(b) The degree to which the system operates as the user expects and provides the services expected.

(2) The PDSS agent uses SPR results to identify problems areas and develop changes that will improve system performance and usability. Additional reviews throughout the deployment and operations phase provide assurance that the SCPs continue to satisfy user needs and improve overall system quality. Content of the reviews is dictated by the initial system corrective actions, problem areas and changes.

b. Emergency changes. In response to critical situations, emergency changes may need to be released to the field within 48 hours. While all changes must undergo validation, verification, and regression testing, emergency changes to deployed systems may not require formal developmental testing or operational testing prior to release. All emergency changes, however, will undergo formal testing with the next planned updates. The PM, with the concurrence of the system user, may only be capable of performing limited testing of emergency software corrections prior to granting release.

c. Test reusability. Test cases, data, and procedures stored in developer SDFs may be necessary or desirable for enabling the LCSEC/PDSS agent to retest software during maintenance more effectively. If so, the appropriate items should be included in the technical data package delivered by the developer.

Table 9–3
Degree of DT/OT needed to support evaluations

Probability	Potential problem impact			
of problem	Negligible	Minor	Major	Catastrophic
Low	Light	Moderate/Light	Moderate/Light	Moderate
Medium	Light	Moderate	Heavy/Moderate	Heavy
High	Moderate	Heavy/Moderate	Heavy	Intnsv/Heavy
Very High	Moderate	Heavy	Intnsv/Heavy	Intensive

Notes:
Intensive: Up to and including full repeated DT/OT from MS III plus changes
Heavy: DT with significant OT
Moderate: DT with OT excursions
Light: DT

Checklist for Determining Potential Problem in Implementing a Software Change Package	Yes	No	Unk
1. Items concerning system performance			
a. Does the software change affect the way the system operates?		x	
b. Does the software change affect the system's operational capability, to include:			
(1) MNS, ORD or operational mission profile?		x	
(2) qualitative and quantitative personnel requirements?		x	
(3) the operational environment?		x	
(4) critical operational issues?		x	
(5) operating procedures?		x	
c. Does the software change affect a critical mission function of the system?		x	
d. Does the change affect safety or security features?		x	
e. Does the change affect the system's critical operational issues and criteria (COIC) or additional operational issues (AOI)?		x	
f. Does the change affect the system's critical technical parameters?		x	
g. Will there be a significant change in the system's throughput? In the throughput of particular components?			x
i. Will there be significant changes to the support software (e.g., operating system, DBMS)?		x	
2. Items concerning interoperability			
a. Does the change affect interfaces with any other systems?			x
b. Is there a change in code for the interface with nondevelopmental (COTS) software?		x	
c. Is there adverse change in system performance caused by execution or management of peripheral devices?		x	
d. Are protocols for communication links affected?		x	
e. Are there changes in the input or output formats?		x	
f. Does the software modification impact other hardware/software interfaces?		x	
g. Will there be changes to procedures for exchanging information with other systems?			
(1) within the battlefield functional area?		x	
(2) with other battlefield functional areas?	x		
(3) with strategic or theater level systems?		x	
(4) with joint systems?		x	
(5) IAW international agreements?		x	
3. Items concerning usability			
a. Is there a significant change in the user displays/reports?		x	
b. Will there be significant changes to the training program?		x	
4. Items concerning system support			
a. Does the change affect the system's support facilities (e.g., software tools, support personnel, support equipment, support documentation)?		x	
b. Does the change affect built-in test equipment?		x	
c. Will there be a change in the organization responsible for PDSS?		x	
d. Does the developer lack experience with the tools or products used to make the change?		x	
5. Items concerning software metrics			
a. Do any requirements remain untested?		x	
b. Were there any catastrophic or major problems (as defined in Table 9-3) experienced during last deployment of the system?		x	
c. Did any catastrophic or major problems occur during any previous testing of this change package? Do any priority 1 or 2 problem reports remain open?		x	
d. Is the number of source lines of code added, deleted, or modified greater than 10% of the total fielded source lines of code?		x	
e. Is the use of computer resources likely to exceed their target upper bound?		x	
f. Have all changed requirements been traced to code and test cases?			x
g. Does the system currently meet its mean time between failure requirements?		x	
h. Is the change package more than 15 percent behind schedule?		x	

Figure 9-1. Example checklist for determining potential problems in implementing a software change package

Chapter 10 Army Software Metrics

Section I General

10–1. Introduction

a. This chapter describes 14 software metrics for gathering information over the life cycle of Army software-intensive systems.

b. Collecting additional metrics is also encouraged to support the unique needs of specific agencies or programs. The Practical Software Measurement initiative provides guidance on selecting other metrics.

c. Many activities contribute to assessing software quality and maturity as described in preceding chapters. The results of those activities, however, cannot all be expressed as uniform, quantitative measurements. Qualitative factors remain important to any software evaluation. Quantitative measures, such as metrics, are less subjective in their interpretation and make changes from previous behavior easier to detect and measure.

d. Software metrics are only one of many factors to consider when evaluating software maturity.

10–2. Policy requirements

a. Previous Army policy required program managers to use and report the first 12 metrics in section II. Recent acquisition reform guidance precludes PMs from requiring developers to use and report a specific set of metrics. The 14 metrics in section II have proved useful in addressing issues integral to managing risk in software-intensive programs. The description of each metric includes a tailoring section with suggestions for alternative implementations. The PMs also have the flexibility to implement each metric to take maximum advantage of the information their software developer has on hand.

b. Department of Defense Directive 5000–2R requires software metrics be used on all acquisition category (ACAT) I, IA, and DOD oversight systems. The cost, schedule, requirements traceability and fault profiles metrics adequately support these reporting requirements. DOD policy also requires those systems to demonstrate, prior to entering dedicated operational testing, that requirements and design are stable and that adequate and sufficient testing of software and interfaces has occurred. The requirements stability, design stability, depth of testing and breadth of testing metrics can serve this purpose.

c. DOD 5000–2R requires MAIS programs to report their progress quarterly (see chap 8). The metrics in this chapter can provide considerable information in preparing these status reports. An illustration applying the Army metrics to each management issue in the report is supplied in section III.

10–3. Types of metrics

a. The Army metrics fall into three general categories as shown in figure 10–1. Management metrics deal with contracting, programmatic, and overall management issues. Requirements metrics pertain to the specification, translation, and volatility of requirements. Quality metrics address testing and other technical characteristics of software products.

b. Software development projects typically track the information and collect the data items needed for the metrics described in this chapter. This detailed data, however, is often used only at lower levels of management within a developer's organization. Summaries are not usually reported to higher level managers in a form suitable to support program management decisions. The suggested metric displays presented in this chapter should be annotated with program specific information. The resulting information displays will provide program managers with the insight needed to make informed decisions on software management issues. Displays other than those suggested may be appropriate depending on the decisions to be made.

c. Several metrics are often needed to evaluate an activity or an issue of interest. For instance, to address whether a program can remain on schedule, relevant metrics include schedule, requirements and design stability, development progress, depth and breadth of testing, and fault profiles. Each metric description includes management information and correlations with other metrics where analysis of program issues takes place.

10–4. Application

The preceding chapters relate the activities of software development to software metrics. Figure 10–2 shows the application of metrics over the life cycle. The same metrics used to monitor the development activities described earlier in this pamphlet are used to monitor the corresponding activities during PDSS.

10–5. Metrics program considerations

In order to gain the most useful insight into software processes and products, the following should be considered when planning a metrics program or when analyzing metrics data:

a. Be sure the metric data definitions are consistent. For example, the definitions for unit, module, function, and lines of code should be established and followed for the project by all involved in collecting and interpreting the metrics.

b. Metric displays should be combined with other qualitative information. Decision makers must consider program issues when analyzing and evaluating metrics data.

c. Metric displays should be used to portray trends over time, rather than placing too much importance on a calculated value at a single point in time.

d. Never use metrics to evaluate personnel. People will focus on manipulating metrics rather than doing their jobs.

e. Metrics can be expensive in terms of resources. Tailor them to use data already available from the software developer. Appendix B discusses how to contract for a software metrics effort.

10–6. Organization and approach

Each software metric is presented in the following format:

a. Description. The type of information the metric is used to assess or present for viewer analysis is briefly described.

b. Application. The typical period(s) in the life cycle when meaningful data are available to support the metric. A reporting frequency is recommended to allow timely trend analysis and to initiate corrective action, if needed. Other information to consider before actually collecting the metric is provided when appropriate.

c. Data definitions. Recommended data items and the level(s) at which it is appropriate to collect the items are listed here. The definitions are elaborated in appendix C.

d. Presentation and analysis. Sample displays are provided for viewing and interpreting the metric data.

e. Management information. Program issues that the metric addresses are discussed in this paragraph, as well as guidance in related areas that may require additional management planning or decisions. Guidance in evaluating this metric with others in this pamphlet to derive a more complete picture of software product maturity is provided.

f. Tailoring. Alternatives for data presentations, reporting frequencies, level of data collection, or other pertinent information regarding the recommended approach are offered.

Section II The Army Metrics Set

10–7. Cost metric

a. Description. The cost metric provides insight into the actual cost expenditures for software development tasks, compared to the initial cost estimates. Data for the metric were selected from the cost accounting system used for most DOD acquisition programs, the cost/schedule control systems criteria (C/SCSC). The C/SCSC, described in DOD Instruction (DODI) 7000.2, is used to track cost, schedule, and technical performance.

Metric	Objective	Measurement
Management Metrics		
Cost	Track planned and actual S/W expenditures	$ spent vs $ allocated
Schedule	Track schedule adherence	Milestone/event slippage
Computer resource utilization (CRU)	Track planned and actual resource use	% resource capacity used
Software engineering environment (SEE)	Quantify developer S/W engineering capability	Overall maturity level and key process area scores
Manpower	Track planned and actual effort and staffing	# labor hours and # personnel
Development progress	Track planned and actual work unit completion	# units designed, coded, integrated
Requirements Metrics		
Requirements traceability	Track reqts to code and test cases	% reqts traced
Requirements stability	Track changes to reqts	
Quality Metrics		
Design stability	Track design changes	Stability index
Complexity	Assess code quality	Complexity indices
Breadth of testing	Track testing of reqts	% reqts tested and % reqts passed
Depth of testing	Track testing of code	% of code and data conditions tested and passed
Fault profiles	Track open vs closed anomalies	# and types of faults, average age of faults
Reliability	Assess system mission failures caused by S/W	MTBF

Figure 10-1. The Army's software metrics

b. Application.

(1) *Data collection and reporting.* Cost data collection begins at the start of the program and continues through fielding and PDSS. The recommended reporting frequency for this metric is at the end of each financial reporting period, which is typically monthly.

(2) *Preparation.*

(a) The first step in applying the software cost metric is to identify the appropriate software work tasks, or activities, as cost elements in a program. Identifying cost elements allows managers to monitor system software cost risk issues. Cost accounting elements are identified through the use of a work breakdown structure (WBS). A WBS element is an identifiable item of hardware, software, services, data or facilities. As a general rule, every development activity in chapters 5 through 8 that applies to a particular program can be classified as a cost element. Procedures for developing a WBS and its hierarchy of levels are defined in Military Handbook (MIL-HDBK) 881. The handbook provides examples of software development activities with definitions of the work performed in each type of activity.

(b) The WBS should allow software costs to be accountable to individual computer software configuration items (CSCIs), builds, subsystems or system as deemed appropriate for the activity and level of visibility needed to monitor risk effectively.

(c) Table 10–1 contains examples of development activities discussed in this pamphlet and collection level recommended for the cost metric. The WBS elements in the table primarily cover technical effort related to producing and maintaining application software.

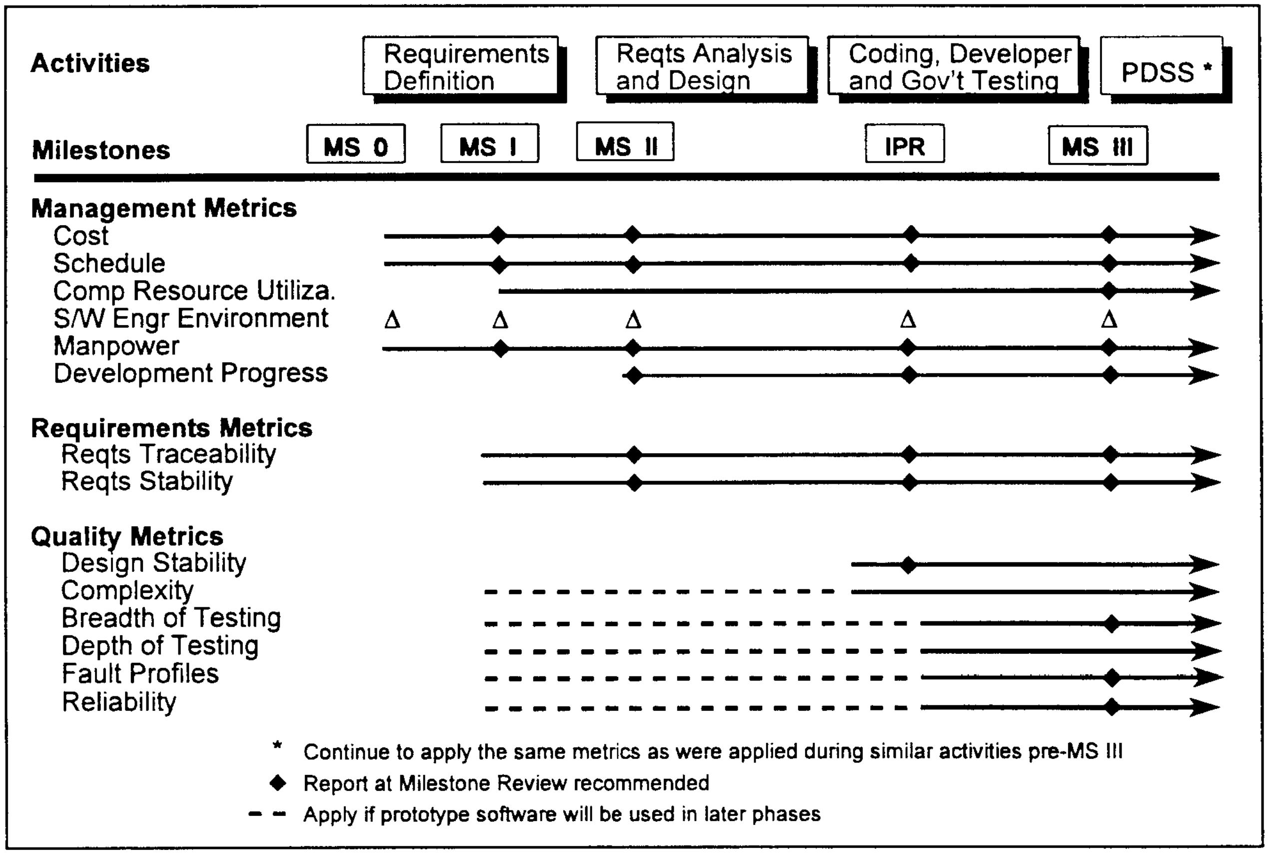


Figure 10-2. Metrics during the life cycle

(d) As appropriate, items dealing with data center facilities, services and hardware, or database administration and maintenance should also be tracked with the cost metric if they are considered potential risks to overall program cost or schedule objectives.

Table 10–1
Examples of software related WBS elements/development activities

By CSCI [1]	By (sub)system
S/W requirements analysis [2]	System requirements analysis
S/W design [2]	System design
S/W implementation & unit testing [2]	S/W development environment [2]
Unit integration & testing [2]	CSCI/HWCI integration & testing [2]
CSCI qualification testing [2]	System qualification testing [2]
Corrective action (S/W PCR resolution) [2, 3]	Project planning & oversight
Software data	Joint reviews
	S/W quality assurance [2]
	S/W configuration management [2]
	S/W product evaluation [2]
	Verification and validation [2]
	S/W fielding [2]
	S/W transition [2]
	Total [2, 4]

Notes:
[1] Separate accounting of CSCIs by build may be appropriate.
[2] Recommended for cost metric tracking.
[3] Includes applicable rework (redesign, recode, unit re-integration and test).
[4] Sum of all software development activities, plus any other software related costs.

c. Data definitions. The following data items are collected for each reported software WBS activity. Identification of the applicable CSCI, build, and system may be needed in addition to the items below.

(1) Budgeted cost of work scheduled: The budgeted cost of work scheduled (BCWS) is the sum of the budgets for all work packages, the level of effort, and apportioned effort scheduled to be accomplished within a given time period.

(2) Budgeted cost of work performed: The budgeted cost of work performed (BCWP) is the sum of the budgets for completed work packages and completed portions of open work packages, plus the applicable portions of the budgets for level of effort and apportioned effort. Some accounting methods refer to BCWP as earned value. Some methods allow earned value for complete activities, while others allow earned value to accrue incrementally.

(3) Actual cost of work performed: The actual cost of work performed (ACWP) is the cost actually incurred in accomplishing the work performed within the given time period.

d. Presentation and analysis.

(1) Figure 10–3 shows a sample graph of cost values plotted over time. Because the cost metric is usually derived from the DOD's C/SCSC data, this is the most common display.

(2) The degree that actual cost values correspond to their original planned values can be calculated and plotted over time as well. A sample graph for the derived values of cost and schedule variances is shown in figure 10–4. Note that a zero value for variances means that the planned budget and schedule were achieved. Cost/schedule performance values, or variances, are—

(a) Cost variance (CV) is the difference between planned and actual cost.

$$CV = BCWP - ACWP$$

(b) Schedule variance (SV) is the difference between the amount of work planned to be performed and actually completed.

$$SV = BCWP - BCWS$$

e. Management information.

(1) Software cost elements may include any expenditures required to develop or maintain a software product. The key to proper application of the cost metric is to identify those WBS elements pertinent to software which pose risk to the overall program.

(2) Exceeding the budget allocation at any point in a program is cause for concern and investigation. This is easily detected as a variance less than zero (for either cost or schedule). Consistently or increasingly negative values for variances indicate that the system may be delivered behind schedule or may exceed the budget.

(3) Cost is associated with all products and activities and can be related to all other metrics. In general, an unfavorable trend in some other metric may adversely affect cost.

(4) The cost metric compares actual software expenditures of work done to the original budget. When assessing overall cost status, however, consider the amount of unfinished work to be done under the remaining budget. Other metrics that show the remaining schedule events, requirements not yet traced and implemented, and number of unresolved software faults provide information about the amount of work remaining. Insight to the risk in achieving software maturity can be derived by estimating the cost of re-work to fix faults and to complete the trace and implementation of requirements to final software products.

(5) Be aware that cost information may arrive as much as 60 to 90 days behind the delivery of other metric data. When evaluating other metrics with cost, be sure that comparable time periods are examined.

f. Tailoring.

(1) The acquirer can stipulate that developers report costs at any level of detail required to effectively monitor the program. Tailoring the cost metric involves developing a WBS that adequately identifies software elements and organizes them at the WBS level most appropriate to expose program risks. The WBS reporting level should satisfy the risk management questions of each individual program. The activities tracked by the cost metric come from the WBS.

(2) Some accounting methods use the cost performance index (CPI) and the schedule performance index (SPI).

$$CPI = BCWP/ACWP \text{ and } SPI = BCWP/BCWS.$$

Values less than one for these indices indicate costs and schedules exceeding estimates, respectively.

(3) A number of commercially produced automated tools are available that can compute BCWS, BCWP, and ACWP based on the accompanying WBS. Using the output of these tools is an effective method of tailoring the cost metric.

10–8. Schedule metric

a. Description. The schedule metric indicates the degree to which program events adhere to a work schedule plan and complements the schedules typically used on programs. This is accomplished by regular reporting of actual achievements in relation to the original schedule. Most importantly, the recommended displays for the schedule metric show the changes to the schedule for future events, over time. Monitoring schedule changes will indicate the level of risk associated with achieving future program milestones and providing key software deliverables on time.

b. Application.

(1) *Data collection and reporting.* Data collection for the schedule metric begins at program start and continues through fielding and PDSS of the system. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* Software development and maintenance programs normally establish a schedule similar to that shown in figure 10–5. Identify any milestone, event or activity which may pose a risk to meeting software or system schedule goals if not initiated or completed on time. Track these events with the schedule metric. It is often advantageous to track events for individual CSCIs and builds separately.

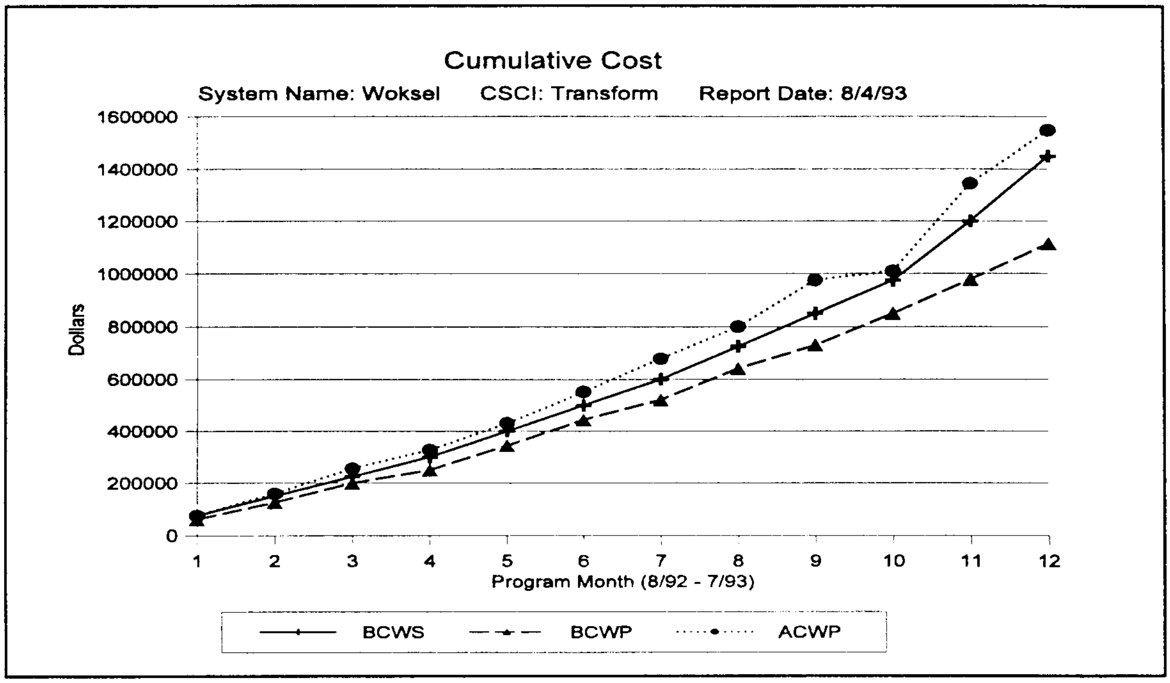


Figure 10-3. Sample cost expenditure graph

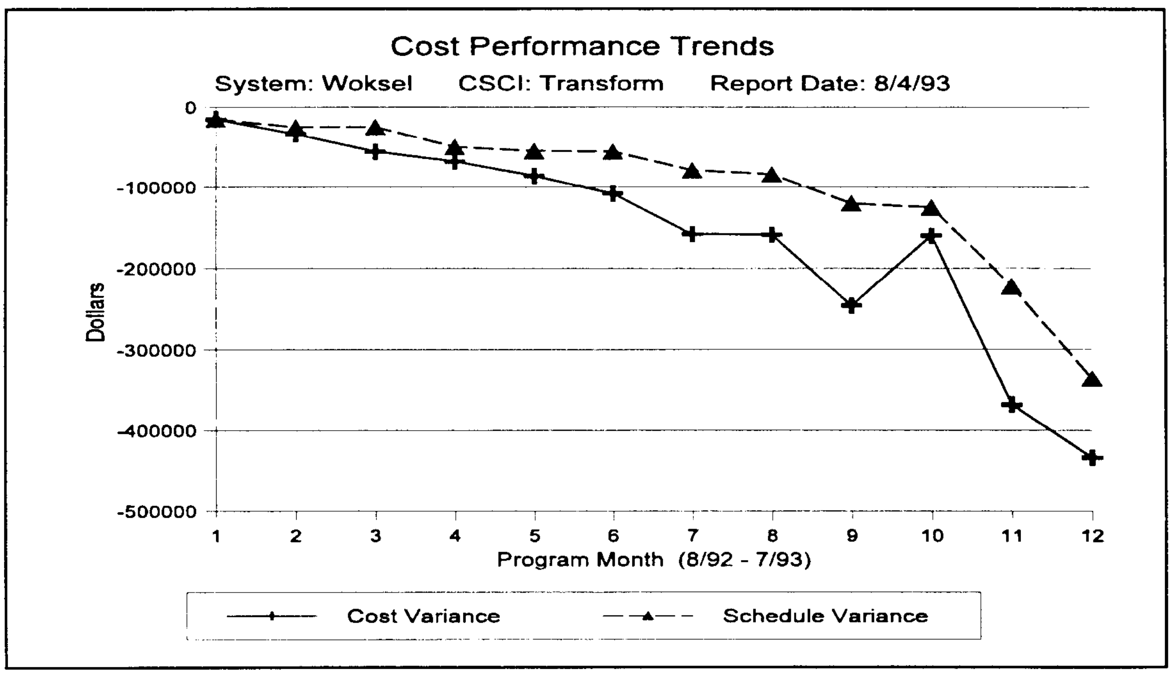


Figure 10-4. Sample cost performance trend graph

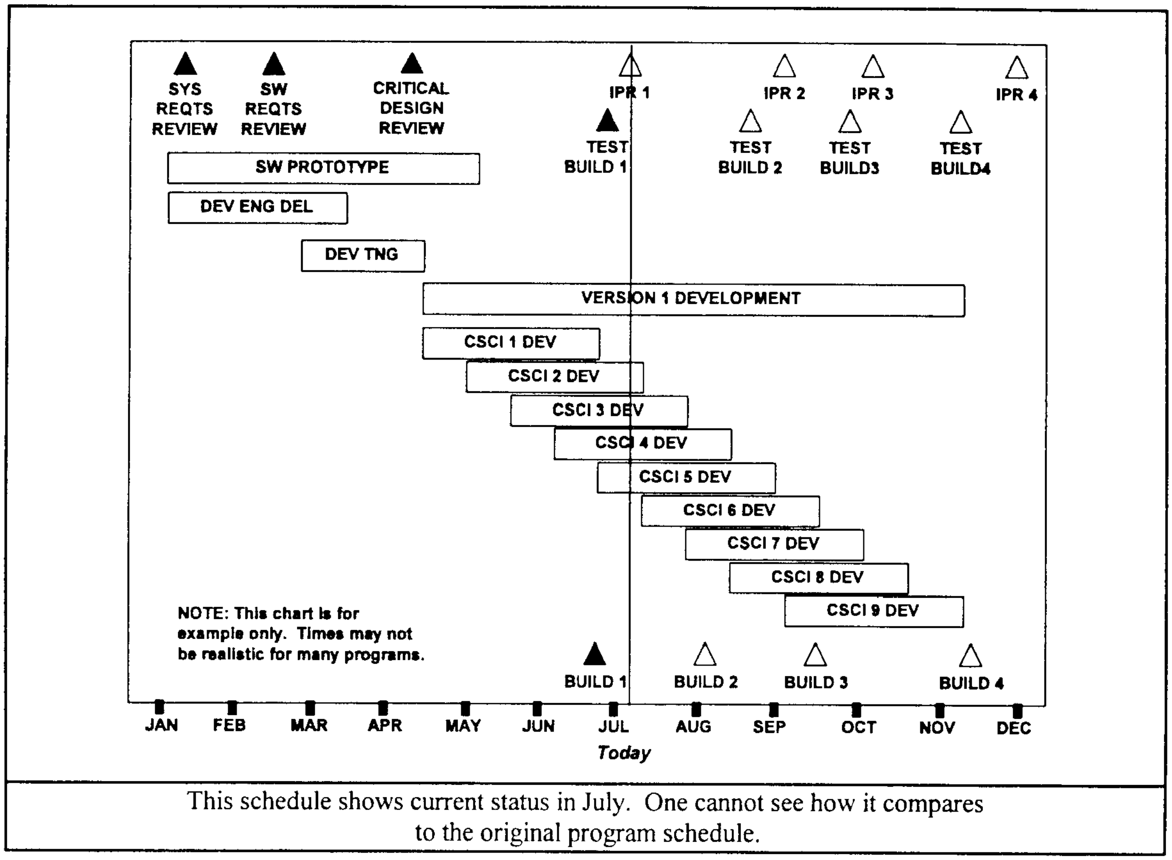


Figure 10-5. Typical program schedule

c. Data definitions. For each item selected for schedule metric monitoring collect—

(1) Planned start date.

(2) Planned end date.

(3) Actual start date.

(4) Actual end date.

d. Presentation and analysis. Information displays for the schedule metric focus on the difference between the original plan and subsequent changes. A problem with many software schedule reporting systems is that new schedules, similar to figure 10–5, are issued to completely replace previous schedules. The result is that information on the original time allocations is lost. Without the ability to compare a new schedule to the original schedule, it is difficult to determine if changes are realistic, or if program risk has been increased.

(1) One recommended display of schedule data is a graph of planned start dates for program milestones and key software deliverables over time, as shown in figure 10–6. Planned start dates for an event are plotted until the event actually begins (that is, an actual start date is reported). In this example, the planned start dates for several software development activities are plotted over the month in which the data was reported. The graph clearly shows the compression of time between the start of software requirements analysis and design. To read the graph, find the metric reporting date (program month) on the x-axis and read the appropriate planned start date on the y-axis. For example, at month one, requirements analysis was planned to start in month two, and the software design was planned to start in month seven. At month two, the start of requirements analysis has slipped to month three (a slip of one month), but the start of software design has remained the same. At month five, the start of requirements analysis has slipped to month six (a total slip of four months), and the software design schedule has slipped only one month.

(2) Another recommended display is shown in figure 10–7. This graph is similar to figure 10–5 but uses bars to represent long-duration events. The bottom of the bar represents the planned start date for an activity; the top of the bar represents the planned end date.

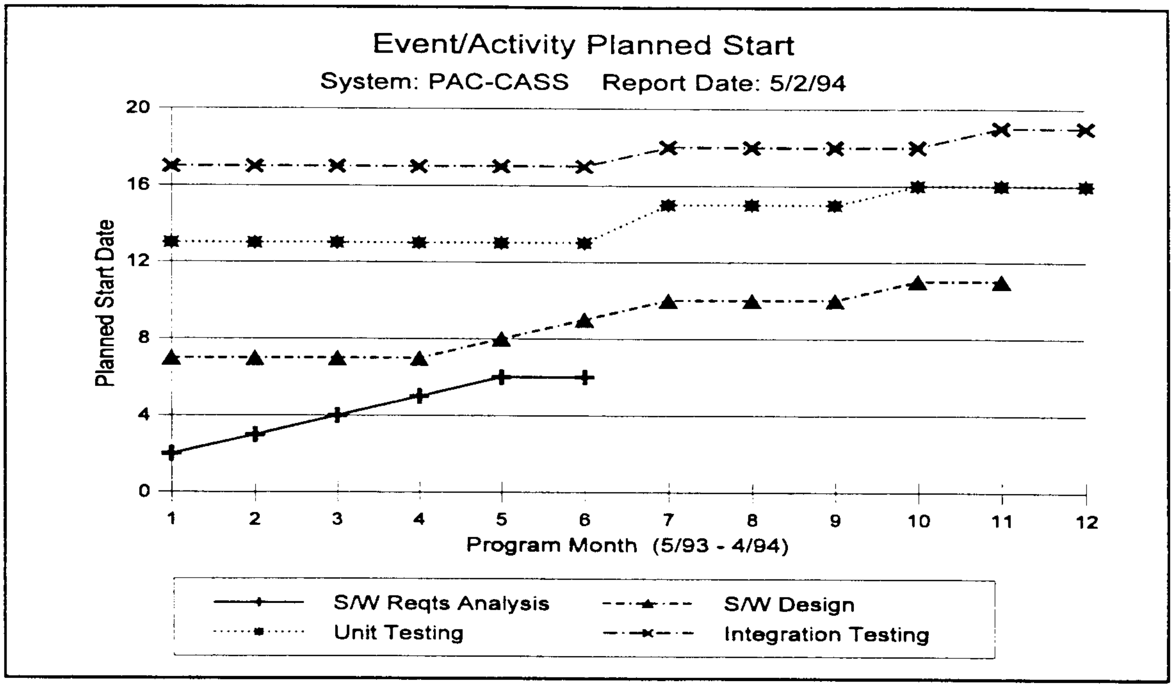


Figure 10-6. Sample schedule metric graph

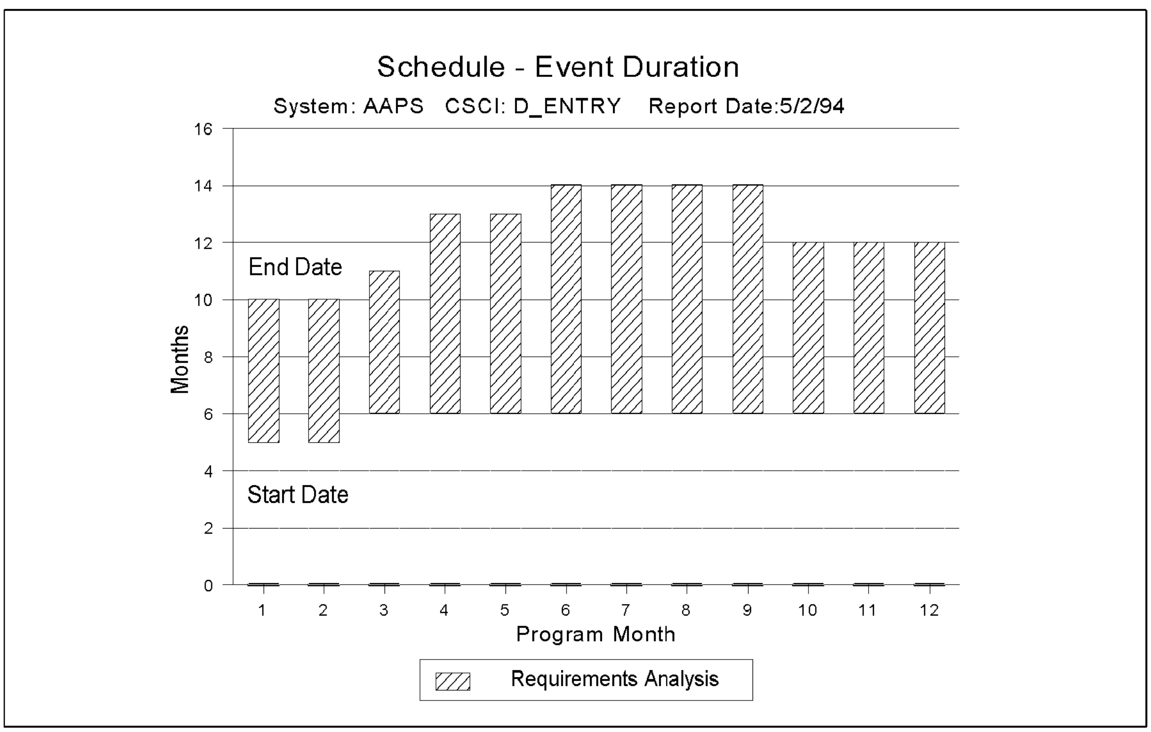


Figure 10-7. Sample graph of changes in activity durations

e. Management information.

(1) The most common problem with a software program schedule is that realistic adjustments of the schedule are not made in response to early changes in program events. Events that do not start on time often do not end on time. In a chain of events, a delay in one event can delay one or more of the events that follow it. Failing to adjust future schedule dates after slipping an earlier event reduces the time and budget to complete future software-related activities. Maintaining the original end dates in a schedule after early delays usually requires additional resources.

(2) The schedule metric indicates potential risks in meeting the schedule for future activities. The recommended displays show clustering or " bunching" of events or whether the amount of time allotted for long-duration events has been shortened or extended. Bunching occurs when schedule slips are experienced in early events and future program events retain their original schedule dates. If bunching is allowed to continue, there may not be enough time to complete all future scheduled events.

(3) The PM may wish to establish criteria for action to be taken in response to schedule changes revealed by the metric. Events that have slipped several times, events whose time allocation is considerably shortened or lengthened, or failure to reschedule future events after significant changes have occurred to early activities may need to be justified by the developer.

(4) Since schedule is associated with all software products and activities, this metric can be correlated with all other metrics. In general, an unfavorable trend in some other metric may adversely affect the schedule.

(5) The schedule metric can be used in conjunction with other metrics to assess program risk. The fault profiles and development progress metrics provide information about the amount of work which remains to be done. The cost metric indicates if resources are available to accelerate the work rate and meet the original schedule. The cost and schedule metrics together can be the best early indicators of problems areas, allowing managers to focus attention in these areas and resolve problems before they get out of hand.

(6) Note that the schedule variance computed under the cost metric does not identify specific events which are behind or ahead of schedule, nor by how much in terms of calendar time. The schedule variance only indicates how many dollars the program is behind or ahead of schedule.

f. Tailoring. It is often easier to portray the schedule metric in a table listing the planned and actual start and end dates for key activities. The graphs, however, present the information in a format which makes the changes to future event allocations much easier to see.

10–9. Computer resource utilization metric

a. Description. This metric shows the degree to which estimates and measurements of computer resources are changing or approaching the limits of resource availability. Constraints in computer resource utilization can lead to poor performance in the operational environment. The primary objective of this metric is to determine whether computer resources are adequate to handle the most demanding anticipated operational workloads. A second objective, no less important, is assurance that reserve capacity for future maintenance and enhancement exists prior to initial fielding.

b. Application.

(1) *Data collection and reporting.* Initial values for computer resource allocations and acceptable limits on utilization are derived from system requirements during the system requirements analysis activity. As system requirements are distributed among hardware and software items as a result of the system requirements analysis, system design, software requirements analysis and software design activities, more refined values are available. The activity descriptions in chapter 5 provide more detail on this process. Collect host (software development/test environment) estimates of resource allocations and projected usage, then the estimates for the target platform. Measurements during the development period should be reported monthly. The CRU measurements should be reported as needed in PDSS, especially when the system configuration is changed, but at least once for each fielded release.

(2) *Preparation.*

(a) This metric requires system computer resources to be identified and usage budgets allocated to them. The minimum set of resources to monitor are central processing units (CPUs), input/output (I/O) channels, storage devices, and memory.

(b) Establish a target upper bound on utilization for each resource. The upper bound is often set to reflect the amount of unused reserve capacity the resource must have to ensure adequate system performance and to provide a margin for growth after delivery. Resource measurements below this value are considered acceptable. Readings above the upper bound are cause for further investigation. The bounds should be set low enough to allow remedial action to be planned and initiated while allowing work in progress to continue in the short term. Target upper bounds for hardware resources are system dependent. A target upper bound is usually set no higher than 50 percent for all resources in MSCR programs (see DA Pam 70–3). An upper bound of 70 percent utilization for AIS may leave adequate reserve. Target upper bounds for all computer resources in a system need not be identical.

(c) In multi-processor environments, each processor should be tracked separately, and each should be allocated a planned utilization. For memory, the resource measured is random access memory (RAM). The RAM for this metric refers to both volatile and nonvolatile (for example, read only) memories. For storage, resources include disk space and other mass storage.

(d) Resource utilization is often measured by the host or target computer system. While the measurements contribute slightly to system overhead, these features typically come with computer systems in their off-the-shelf configuration. In instances where the system does not measure itself in terms of resource utilization, the utilization should be measured using probes or other similar devices.

(e) CRU should be measured when the system is operated in accordance with the operational mode summary/mission profile during development, typically during qualification testing, DT, and OT.

c. Data definitions.

(1) For each computer hardware resource in the system collect—

(a) A resource identifier.

(b) Type of resource.

(c) Units of measurement.

(d) Maximum capacity of resource.

(e) Target upper bound on resource usage (percent of capacity).

(f) Projected resource usage (percent of capacity).

(g) Actual resource usage (percent of capacity).

(2) It is recommended to track at least the hardware resources of CPUs, I/O channels, storage devices, and RAM.

(3) For each CSCI in the system collect—

(a) CSCI identifier.

(b) Items b through g of paragraph 10–9 *c*(1) above for the CSCI's allocation and use of a particular hardware resource.

(4) It is recommended to track at least the RAM and mass storage used by each CSCI.

(5) Actual usage should be measured during peak operational periods and should include the operating system and other off-the-shelf software as well as the software being developed. Similarly, projected usage should be based on estimates of peak operational periods and should include the off-the-shelf software and the software under development. Peak load conditions for computer resources usually occur under the operating environment and circumstances described in the OMS/MP.

d. Presentation and analysis.

(1) Figure 10–8 is a sample graph of utilization values for a single CPU resource plotted over time. Similar graphs can be constructed for the utilization of all other CPUs plus each I/O and memory resource.

(2) As software development proceeds, the measured values for each category should be projected for the operational environment. For instance, stimulation testing of system functions yields CRU values that can be used to project the CRU for functions yet to be developed.

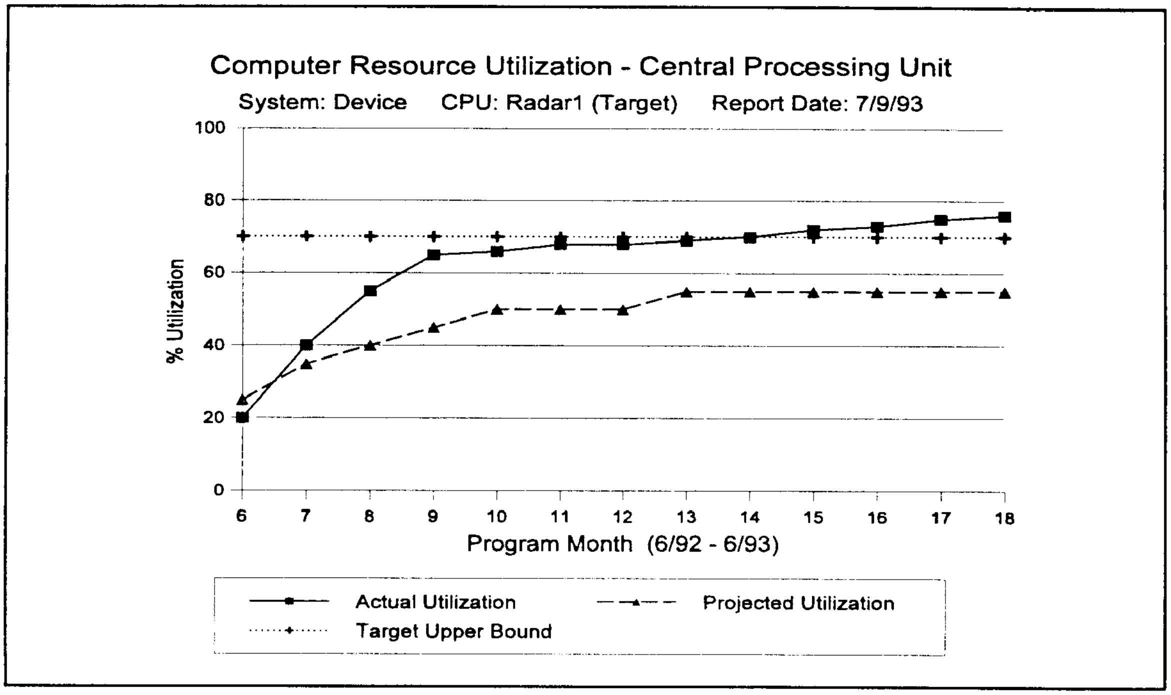


Figure 10-8. Sample computer resource utilization graph

e. Management information.

(1) Resource utilization tends to increase during system development. Therefore, adequate planning is essential to ensure that the software operation does not put undue demands on the target hardware capabilities. This metric tracks utilization over time to make sure that sufficient capacity remains for future growth and for periods of high stress loading.

(2) This metric can be applied to a system architecture which is distributed or centralized.

(3) When target upper bounds are approached or exceeded, it may be necessary to change hardware, change software, reallocate requirements, or raise the target upper bounds. The last option should not be pursued unless the original target upper bound was well below the usable capacity of the resource.

(4) A target upper bound above 90 percent is usually too high to provide sufficient early warning of potential resource overload.

(5) CRU reporting may reveal under-utilized computer resources. Unexpectedly low utilization values may be a result of initial overestimation, reduction of functionality due to requirements changes, reallocation of functions to other computer resources during design and implementation, or measurements taken at less than peak load conditions.

(6) The developer should be tracking or projecting overall system-level performance using measures such as messages per hour, transactions per minute, or targets per engagement. These measures should also be taken when the system is operated in accordance with the OMS/MP. If the system is not meeting these system-level requirements, CRU of the various computer resource components should be examined to determine where the performance bottlenecks are.

(7) Alleviating one resource bottleneck sometimes exposes others. When the first problem is remedied, the device's throughput may improve to such an extent that previously untaxed resources become overloaded.

(8) In instances where the development testing and target environments differ in types and/or capacities, caution should be taken in computing and analyzing the measures. Projections based on the development test environment are acceptable up to a certain point, but testing on the target hardware should take place as early as possible.

(9) Growth in resource utilization is often not predictably linear. Also, utilization may not always be directly controllable by the developer. For example, upgrading commercial NDI software to newer releases may reduce the resources available to application programs due to enhancements added by the vendor. Incremental developments or other management actions that defer requirements should take this into account when preparing resource budgets.

(10) Utilization problems in CPU and memory are often observed as " deadlocks,"" crashes,"" high traffic," and " slow response."

(11) CRU relation with other metrics is summarized in table 10–2.

Table 10–2
CRU relation with other metrics

Metric	Relation
Cost	Potential cost overruns due to additional hardware, software redesign, etc.
Schedule	Potential schedule slips due to redesign or reallocation.
Requirements traceability	Have all requirements been implemented in the code? What is the utilization impact of those that have not?
Requirements stability	Utilization values change as requirements/code change.
Design stability	Utilization values change as code changes.
Fault profile	Over-utilization can lead to serious faults.
Reliability	Over-utilization can lead to system failures.

f. Tailoring.

(1) The recommended CRU reporting frequency can be modified from monthly to occur less often; perhaps only at specified points, bimonthly or quarterly. However, CRU is difficult to correct after

thresholds have been exceeded and early warning through CE may be beneficial.

(2) CRU could also be tracked for implementation and test resources in the software development environment that are not part of the deliverable system, such as workstation utilization or availability. This will ensure developers are getting the computer resources to perform their jobs.

(3) CRU metric data gathering can be a part of the regular operating system functions, or passive monitors may be implemented.

(4) Tailoring may be appropriate for situations when dynamic allocation, virtual memory, parallel processing, multitasking, or multiuser-based features are employed.

(5) The developer can collect additional metrics when and where appropriate. These may include tracking the amount of swapping, paging, and network utilization as well as response time.

10–10. Software engineering environment (SEE) metric

a. Description. The software engineering environment metric provides a rating of the developer's application of software engineering principles. Examples of these principles are the use of structured design techniques, the extent of tool usage, and the use of requirements management techniques. If practical, assessments can also be applied to materiel developer personnel or the program manager's matrix support staff to assess their capabilities with respect to developing software. Rating of software developers should be performed by a qualified independent group. Performing a software engineering environment assessment is described in reports Carnegie Mellon University/Software Engineering Institute (CMU/SEI) –87–TR–23 and CMU/SEI–93–TR–24. The acquirer is responsible for ensuring that the proper methodology is used during the assessment of software process maturity.

b. Application.

(1) *Data collection and reporting.* An assessment should occur whenever a software developer is brought onto a program. Reassess any previously identified risks at major milestones to monitor improvements in the developer's software engineering process maturity.

(2) *Preparation.*

(a) The SEE metric uses the Software Engineering Institute's (SEI's) capability maturity model (CMM) to measure a developer's efforts in identifying and improving software process maturity indicators. The CMM provides an organized strategy for process improvement in stages. The stages are an evolutionary path, in that improvements at each stage form the foundations for the next stage. Information on an organization's current process is gathered primarily by means of questionnaires.

(b) The SEI model outlines five levels of process maturity. Each level, except level 1, is composed of constituent parts. Each level focuses attention on a different set of process attributes, also called key process areas. Each key process area (KPA) consists of numerous key practices, that when addressed collectively, accomplish the goals of the KPA. Some of the key practices are selected as key indicators of whether the goals of a key process area are accomplished. Questions in the CMM questionnaires are based on these key indicators.

(c) Have an assessment team follow the methodology outlined in CMU/SEI–93–TR–24, which includes the following:

1. Collect questionnaire data from developer.

2. Conduct follow-up visits to answer further questions, observe tools, and so forth.

3. Compile the findings from the previous two steps to identify software process strengths and weaknesses in key process areas.

4. Calculate the developer's overall process maturity level. A particular level is achieved when findings indicate all key process areas for that level and every level below it are fully satisfied. Table 10–3 shows the CMM's allocation of KPAs to maturity levels.

c. Data definitions. Data for the SEE metric comes from the findings of a process maturity assessment. For each assessment collect—

(1) Developer name or other identification.

(2) The process maturity level determined as a result of the assessment.

(3) The date the level was assigned.

(4) The type of assessment performed, such as developer's self-assessment, acquirer's assessment of the developer, or independent third party assessment.

(5) For each KPA that was examined record the item's assessment outcome (for example, satisfactory, unsatisfactory, or not reviewed).

Table 10–3
Capability maturity model definitions

At this process maturity level	An organization demonstrates adequate capability in these key process areas
1 (initial)	
2 (repeatable)	Software configuration management
	Software quality assurance
	Software subcontract management
	Software project tracking and oversight
	Software project planning
	Requirements management
3 (defined)	Peer reviews
	Intergroup coordination
	Software product engineering
	Integrated software management
	Training program
	Organization process definition
	Organization process focus
4 (managed)	Quality management
	Process measurement and analysis
5 (optimizing)	Process change management
	Technology innovation
	Defect prevention

d. Presentation and analysis. Simple tabular presentations are usually adequate to easily determine a developer's past and current process maturity level(s), KPAs that need improvement and those areas in which the developer demonstrates strength.

e. Management information.

(1) The software engineering environment rating provides a consistent measure of the capability of a developer to use modern software engineering techniques in their development process, and therefore their capability to instill such principles and characteristics in their products. The basic assumption to this approach is that a quality process results in a quality product. Other metrics and evaluation techniques should be used to examine product quality.

(2) Although software engineers and managers often know their problems in great detail, they often disagree on which improvements are most important. The SEE metric's use of standard CMM questionnaires allows engineers and managers to focus on a limited set of key processes and work aggressively toward implementing them, rather than being overwhelmed by the total process.

(3) The SEE rating assists the acquirer in identifying and narrowing risk to specific areas generally accepted to have an affect on effective software production. The PM should use the SEE metric to focus on determining developer capabilities and to gauge the ability and willingness of the developer to improve in weak areas over time, not just on selecting one developer over another.

(4) An assessment often reveals that a developer is proficient in KPAs from one or more CMM level higher than the rating number assigned. For that reason, more information than the maturity level number is relevant to appraising actual capability.

(5) SEE assessments conducted by an SEI trained team are desirable. However, acquirers are encouraged to train their staff how to determine software development capability and to perform informal assessments themselves.

(6) The software engineering environment rating can be used by developers to find and improve weaknesses in their software development process on their own.

(7) Be aware that the SEE metric reflects current practices only

at the time of the assessment for the particular organization examined. Changes in a developer's corporate environment, management philosophy or other factors may lead to circumstances that detrimentally affect KPAs over time. Therefore, occasional informal re-examination of KPAs previously judged satisfactory may be appropriate.

(8) A higher SEE rating should have a positive impact on all other metrics.

f. Tailoring.

(1) After an initial SEE evaluation, assessments can be tailored to focus primarily on risk areas uncovered in previous assessments.

(2) After an initial rating, a developer should periodically review progress in the key process areas that were lacking in the prior evaluation. Reporting at each major milestone is only beneficial for long duration programs since it takes time to implement changes and see improvement in corporate policy and procedure, and ultimately software products.

10–11. Requirements traceability metric

a. Description. The requirements traceability metric measures the level to which software products have implemented requirements allocated from higher level specifications. Software products include specifications, software design, code, and test cases.

b. Application.

(1) *Data collection and reporting.* Tracing software requirements begins with the first specification produced in response to a defined mission requirement for which an automated information solution is foreseen and continues throughout the life of the program. The recommended reporting frequency for this metric is at major milestones during system or software development, or at major software release points during PDSS.

(2) *Preparation.*

(a) The requirements traceability metric is an accounting of how many requirements from one document are addressed in other documents. This is typically from higher to lower levels of specification and corresponds to the evolution of requirements and design depicted on the left side of figure 1–2. Requirements are also tracked to the tests that verify them; the right side of figure 1–2. In order to do this, the hierarchy of technical documentation must be determined, and the relationship between the requirements in the different documents evaluated.

(b) Top-level requirements for DOD systems are defined in the mission needs statement (MNS). This original statement of user requirements is expanded in subsequent documents. The specific documentation set developed for a software-intensive system is tailored to its acquisition category and other program-specific issues. Table 10–4 is a representative matrix of different levels of requirements to the typical documents in which they are specified, elaborated or verified. The list refers only to documents described elsewhere in this pamphlet; other applicable documentation may be substituted or added.

(c) At least one document from each level should be tracked with the requirements traceability metric. The important point is that a logical series of software requirements can be followed from top-level operational requirements down to code and test cases. The term " software requirements" for this metric includes any software interface requirements.

(d) The act of tracing means determining whether the requirements in one document are addressed in another. Usually the trace is forward, from higher to lower level of detail. The document with the lower level of detail was usually based on the higher level document. The objective is to verify that all higher level requirements are allocated to lower levels and that lower levels do not add any new requirements. The latter is often more easily accomplished by reversing trace direction from lower to higher levels, also called a backward trace.

(e) The relationships in table 10–5, or their equivalents, are recommended for tracking with this metric.

(f) The primary measure collected in the requirements traceability metric is the account of requirements successfully traced from one level to another. To facilitate this calculation a software requirements traceability matrix (SRTM) can be used to collect and organize the summary data described in the previous steps. The SRTM should contain enough information to allow assessment of the relationship between various levels of requirements and the requirements to their design and test cases. An example of a SRTM is shown in figure 10–9. The SRTM identifies the document(s) selected for tracing at each level in columns and their common requirements in rows representing the links from top-level system requirements to detailed software requirements. In this format, identifying units which represent a required system function should be apparent. Question marks in the sample SRTM mean that tracing has not yet occurred. The degree of completion of the SRTM depends upon the current stage of the software life cycle and which documents are available for review. Also note that at each level of the trace, a single requirement can be traced to multiple lower level requirements.

c. Data definitions.

(1) For each system or software documentation level tracked to another, collect—

(a) Names of the two documents assessed.

(b) Number of system/software requirements in the" traced from" document.

(c) Number of requirements in " traced from" document successfully traced to the " traced to" document.

(d) Number of requirements in " traced from" document that could not be traced to the " traced to" document.

(e) If a backward trace is also performed between the two documents, record the number of requirements in the " traced to" document that were successfully traced back to the" from document," and the number of requirements in the " traced to" document that could not be successfully traced back to the " from document."

(2) It is suggested that software requirements be reported by individual CSCI.

(3) The relationships in table 10–5 are recommended for tracking with this metric.

d. Presentation and analysis. Figure 10–10 is an example graph recommended for the requirements traceability metric. The chart provides a summary of a CSCI's software requirements traced forward to lower levels of design and code and backward to system requirements.

e. Management information.

(1) Software test management procedures dictate that software requirements should be traced to their individual qualification test cases. Recording this trace provides visibility to ensure that software requirements are adequately tested.

(2) Requirements traceability aids in determining the operational impact of software problems. Failed requirements can be tracked back to specific mission needs.

(3) Due to the detailed nature of the requirements traceability metric, collecting this data is most cost effective if it is a normal product of software development or a V&V effort. The consolidated software standards identified in chapter 2 and table 10–5 request forward and/or backward traces be provided as part of many of their software documentation products. The SRTM should be part of the developer's deliverable technical data package.

(4) The SRTM is normally prepared by the software developer, but should also be verified by an independent organization, such as an IV&V agent or LCSEC/PDSS personnel prior to software transition.

Table 10–4
Sample requirements level to technical document correlation

This type of requirement	Is typically elaborated/implemented in	And verified by tests described in
User/mission	Mission Needs Statement (MNS) Operational Requirements Document (ORD)	Test and Evaluation Master Plan (TEMP), System Evaluation Plan (SEP)
User/system	Users' Functional Description (UFD) Operational Concept Description (OCD)	TEMP, SEP
System	System/Subsystem Specification (SSS) Interface Requirements Specification (IRS) [1]	Detailed Test Plan (DTP), Software Test Plan (STP) [1]
Software design	Software Design Description (SDD), Interface Design Description (IDD), Database Design Description (DBDD)	Software Development Files (SDFs)
Unit design	SDD, IDD, DBDD, Software Product Specification (SPS) [2]	SDFs

Notes:

[1] IRS and STP apply when the system is an information system.

[2] SPS contains or references the code and data.

[3] Software documentation based on MIL-STD498/IEEE Std P1498/EIA IS 640.

[4] All documents not applicable to all programs.

Table 10–5
Recommended items for requirements traceability metric tracking

From	To	Backward trace also
User requirements (mission) (MNS/ORD)	User requirements (system) (UFD/OCD)	Yes
User requirements (system) (UFD/OCD)	System requirements (SSS)	Yes
System requirements (SSS)	Software requirements (SRS, IRS)	Yes
Software requirements (SRS, IRS)	Software design high level (SDD) [1]	Optional
Software requirements (SRS, IRS)	Software design detailed (SDD) [2]	Optional
Software requirements (SRS, IRS)	Code (SPS)	Optional
Software requirements (SRS, IRS)	Software qualification test cases	Optional

Notes:

[1] CSCI level design.

[2] Unit level design.

(5) The PM and user representative may also want to evaluate the SRTM. This evaluation can be intensive in time and effort but worth the cost when problems or discrepancies are discovered and corrected early.

(6) When evaluating the SRTM, consider the criticality of the requirement to the system user and the criticality of the resultant software function to system operation. A formal method may be used to identify requirements which address key user operations or critical system functions. Another method is to identify those units which appear most often in the SRTM. These units represent crucial basic software function because they are needed for multiple system requirements and functions. These units can be developed earlier and be given increased test scrutiny.

(7) Incremental or evolutionary acquisition strategies, such as rapid prototyping , where all requirements are not known in advance or specified to the same degree of detail, require the trace of requirements be an iterative process. As new requirements add more functionality to the system, the SRTM is revised and augmented.

(8) The SRTM can be a valuable management support tool at system requirement, design or other joint reviews. It may also indicate those areas of software requirements or design which have not been properly defined.

(9) The PM should establish criteria for requirements traceability thresholds for proceeding from one activity to the next, for example, the percentage of SRS requirements which need to be traced to detailed design before start of coding. Required levels of traceability should be based on the degree of risk assumed for requirements that are not traceable to this point. Individual thresholds are system specific.

(10) During PDSS, if a function is modified, the SRTM can be used to focus regression testing on particular CSCIs/units.

(11) This metric does not provide information on whether tests have been executed or report test success or failure of specific requirements. The SRTM can be tailored to include test result status if desired.

(12) The relation of requirements traceability with other metrics is summarized in table 10–6.

f. Tailoring.

(1) Implementing the requirements traceability metric in the manner described above is complicated by non-hierarchical implementations, such as object-oriented techniques. Program-specific implementation guidance should be established for tracing and counting requirements using object-oriented requirements analysis and design methods. As a minimum, the percent traceable from the SRS to code and SRS to test cases should be provided.

Mission Need	Oper. Concept	System Reqt.	Software/ Interface Reqt.	Software/ Interface Design	CSCI	Unit Design	Code	Software Test Case
MNS	OCD	SSS Paragraph Number	SRS/IRS	SDD/IDD	ID #	CSCI#Unit#, Function	Function, SLOC	STD Par. Number
3.0	3.1	3.1	SRS 3.1.2	SDD 3.1.7	01	01004, Compare	Compare, 141-218	3.1.1 to 3.12.4
		3.1.3	SRS 3.14.2 IRS 3.17.1	SDD 3.10.6 IDD 3.11.2	01	01013, Add	Add, 417-509	3.10.1 to 3.12
		3.1.6	SRS 3.22.2, 3.22.3	SDD 3.15.3, 3.16.7	03	03009, Sort	Sort, 738-822	3.15.2 to 3.15.8
	3.2	3.2.5	SRS 3.2.16	SDD 3.2.2	01	01006, Send	Send, 418-492	3.2.7
4.0	4.1	4.1	SRS 4.1.2	SDD 4.1.3, 4.1.9	01	01003, Divide 01004, Increase 01011, Relate	Divide, 161-248 Increase, 361-438 Relate, 861-948	4.1.3, 4.1.8, 4.2.9
	4.1	4.1.5	SRS 4.1.5 IRS 4.1.9	SDD 4.1.8 IDD 4.11.2	01	01007, Repeat	Repeat, 117-209	4.1.5
	4.2	4.2.2	SRS 4.2.5 IRS 4.2.9, 4.2.11	SDD 4.2.4 IDD 4.2.11	02	02013, Expand 02014, Amplify	Expand, 738-822 Amplify, 1061-1148	4.2.7, 4.2.13, 4.2.14, 4.2.16
	4.3	4.3.3	SRS 4.3.5, 4.3.8	SDD 4.3.3	04	04003, Equate	Equate, 418-492	4.3.3
	4.3	4.3.3	SRS 4.3.6	SDD 4.3.9, 4.3.11	04	?	?	?
	4.3	4.3.4	IRS 4.3.12, 4.3.17	IDD 4.3.17, 4.3.19	03	03016, Total 03019, Append 03022, Match	Total, 1461-1548 Append, 1761-1838 Match, 2161-224	4.3.33, 4.3.40, 4.3.41, 4.3.45
	4.4	4.4.1	SRS 4.4.6	SDD 4.4.4	01	01001, Relay	Relay, 2761-2848	4.4.6, 4.4.9

Figure 10-9. Example of a software requirements traceability matrix

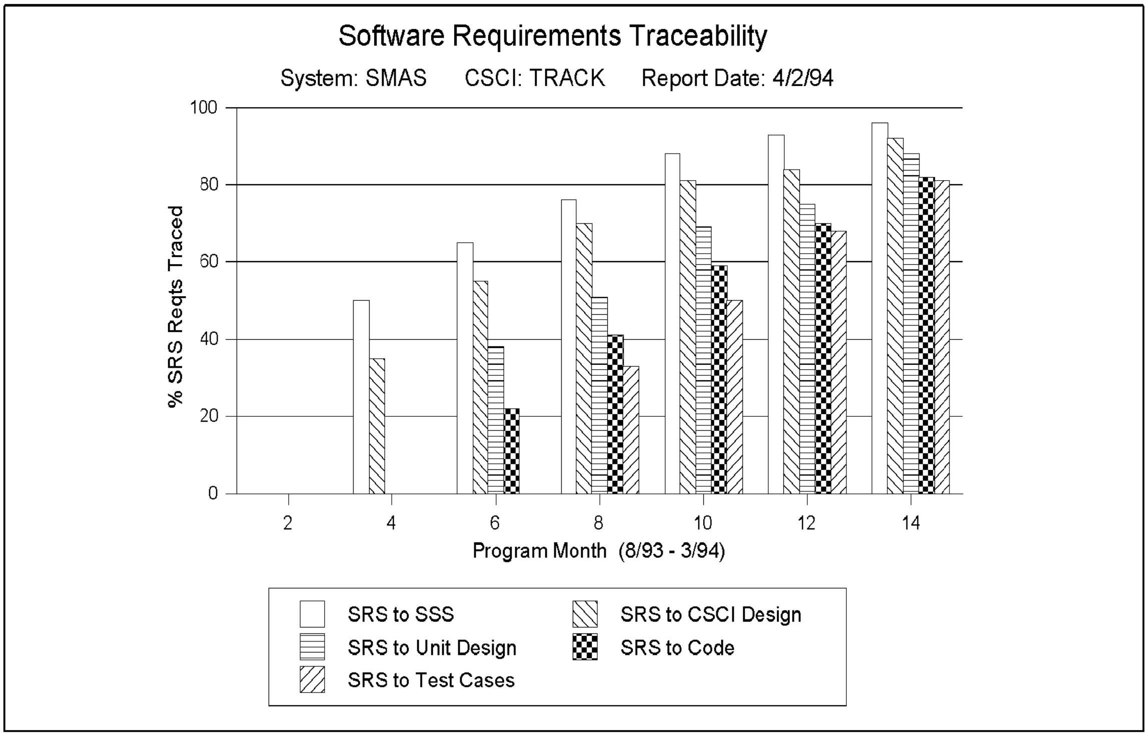


Figure 10-10. Sample requirements traceability graph

(2) The recommended level of reporting software requirements traceability data is at the CSCI level. Lower levels of reporting, such as against specific software units, may be appropriate to focus on risk areas.

Table 10–6
Requirements traceability relation with other metrics

Metric	Relation
CRU	As more requirements are implemented in the design and code, utilization values will increase. Are any target upper bounds being approached or capacities exceeded?
SEE	A developer with a higher maturity level rating can be expected to have an institutionalized process to ensure all requirements are implemented and tested.
Requirements stability	Requirement changes should be reflected in the documented design and code. Have changes been traced to design, code, and test cases?
Design stability	Requirements which are not implemented (or traced) to design description documents (SDD, IDD, and DBDD) prior to detailed design may entail design changes when they are later implemented.
Breadth of testing	Have test cases been developed for each requirement?
Fault profile	A large number of software problem/change reports in the requirements category may indicate inadequate traceability of requirements.
Reliability	Many software “ failures” are actually the result of system or user requirements which are not found or implemented improperly in the final software code.

(3) It can be worthwhile to perform additional backward traces on items marked optional in table 10–5, such as from code to detailed design requirements. This can be accomplished, for example, by making a list of all the distinct units which appear in the“ code” column of the SRTM. The list is then compared with the total list of units in the detailed design documentation or alternate representation, such as automated flow charts. Any unit which appears in the “ code” column, but is not found in the detailed design, may not support a requirement. The actual functions and need for these units should be investigated.

(4) The SRTM approach can be applied to other types of requirements traces as well as software. The baseline correlation matrix described in DA Pam 73–5, for example, documents major system requirements, operational requirements (MNS, ORD), COICs and system measures of effectiveness and performance to ensure requirements remain consistent and is used as a tool in developing additional operational issues (AOIs) and their associated measures.

10–12. Requirements stability metric

a. Description. The requirements stability metric indicates the degree to which changes in the software requirements or changes in the developer’s understanding of the requirements are affecting the development effort. It also allows for determining the cause and source of requirements changes.

b. Application.

(1) *Data collection and reporting.* Collection can begin with approval of the mission need statement. Collection begins in earnest during the system requirements analysis activity and continues for the lifetime of the system. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* Mechanisms to perform and report product evaluations, a corrective action system and configuration management procedures need to be in place in order to collect the data defined below. In order to monitor the source of requirements changes, it is recommended that some data items be collected separately for user and developer categories.

c. Data definitions. Collect for each CSCI—

(1) Software requirements discrepancy status (cumulative total detected and cumulative total resolved).

(2) Total number of source lines of code (SLOC).

(3) Total number of SRS requirements.

(4) Number of SRS requirements added due to approved engineering change proposals - software (ECP-Ss).

(5) Number of SRS requirements modified due to approved ECP-Ss.

(6) Number of SRS requirements deleted due to approved ECP-Ss.

(7) Number of SLOC affected by approved ECP-Ss (proposed by user/proposed by developer).

(8) Number of software units affected by approved ECP-Ss (proposed by user/proposed by developer).

(9) Number of ECP-Ss generated from requirements changes (proposed by the user/proposed by the developer).

d. Presentation and analysis. Figure 10–11 shows cumulative requirements discrepancy counts, detected and closed, over time. Figure 10–12 shows the number of ECP-Ss submitted each reporting period by both the user and the developer.

e. Management information.

(1) When a program begins, the details of its operation and design are rarely complete, so it is normal to experience changes in the specifications as the requirements become better defined over time. (Note: Prototyping can help alleviate this problem, or at least cause refinement to happen earlier in development.) When technical reviews reveal inconsistencies, discrepancy reports are generated. Modifying the design or the requirements to alleviate a problem results in closing the associated discrepancy report. When a change is required that increases the scope of the system, an ECP-S is submitted.

(2) Allowances should be made for lower requirements stability early on in cases where prototyping is used. At some point, however, the requirements should be firm so that only design and implementation issues will cause further changes to the specifications.

(3) The plot of open discrepancies can be expected to spike upward at each review and to diminish thereafter as the discrepancies are closed. High requirements stability is indicated when the cumulative discrepancies curve levels off with most discrepancies having reached closure.

(4) For each engineering change, the amount of software affected should be reported in order to track the degree to which ECP-Ss increase the difficulty of the development effort. Only those ECP-Ss approved by the configuration control board should be tracked.

(5) It is recognized that the amount of SLOC is somewhat dependent on both the application language as well as programmer style. The key is to watch for significant changes to SLOC due to requirements changes.

(6) The PM should establish criteria for requirements stability thresholds for proceeding from one activity to the next. For example, after joint technical review of the software requirements, the requirements should be stable enough to allow the design to be converted into code.

(7) The PM should also establish criteria for time to close open requirements discrepancies. Cost and schedule impacts may be noted when requirements discrepancies remain open after 30 days.

(8) Causes of program turbulence can be investigated by looking at requirements stability and design stability together. If design stability is low and requirements stability is high, the transfer from design to code is not working well. If design stability is high and requirements stability is low, the transfer from the users to the design activity is not working well. If both design stability and requirements stability are low, neither process is working well.

(9) The relation of requirements stability with other metrics is shown in table 10–7.

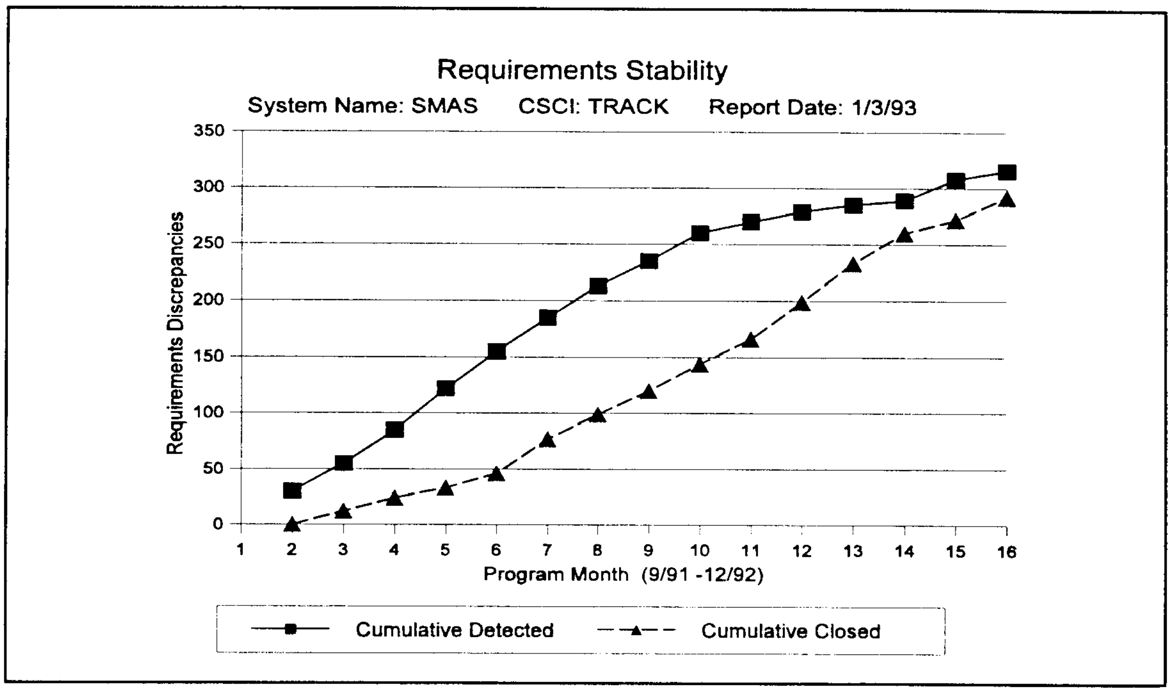


Figure 10-11. Sample graph of requirements discrepancies over time

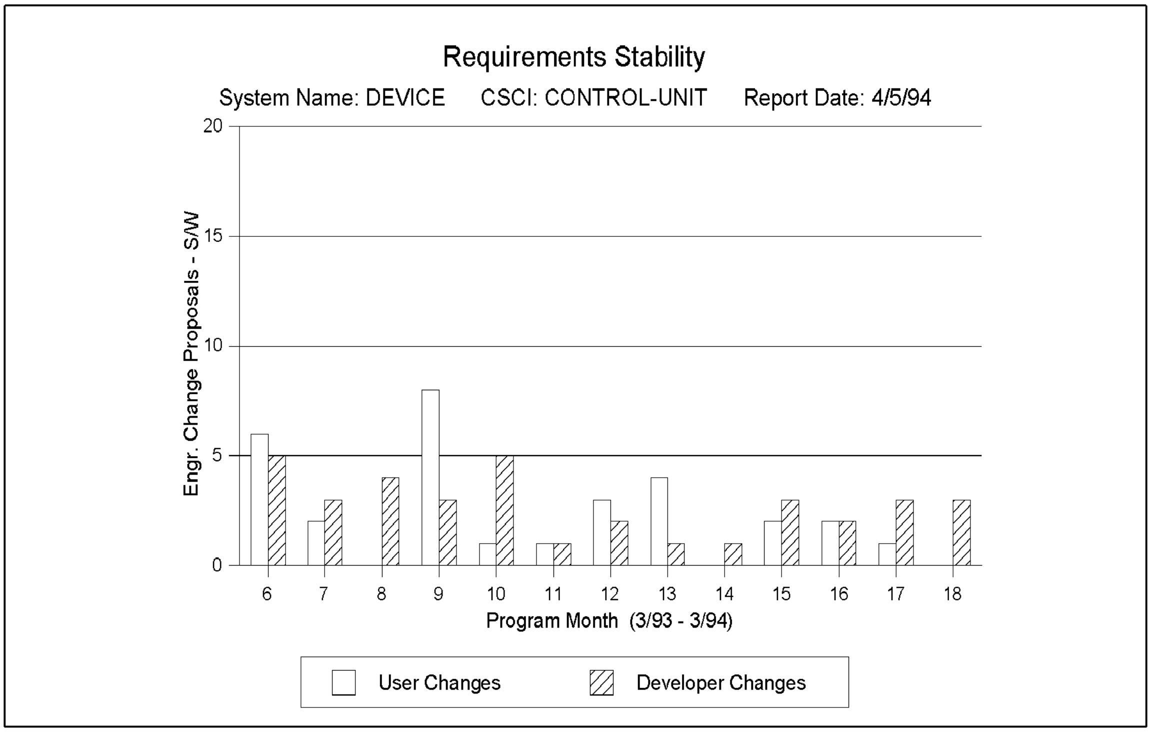


Figure 10-12. Sample graph of ECP-Ss over time

Table 10–7
Requirements stability relation with other metrics

Metric	Relation
Cost, Schedule	Cost and schedule will be adversely affected by an unusually large number of requirements changes. The later in the life cycle requirements changes occur, the greater the severity.
CRU	Changes to functionality may require more resources or cause allocations to be redistributed. Have requirements changes increased utilization measures?
Requirements traceability	Ensure that requirements changes are traced to design, code, and test cases. Requirements that trace to many design elements will take more time and effort to modify and test.
Design stability	As requirements change, software units are modified, added, and deleted. Expect design " instability" as a result of changing requirements. Design stability not changing when requirements are changing indicates a problem.
Breadth of testing	Test cases should be run for changed/new requirements. Does breadth of testing data reflect the changed/new requirements?
Development progress	Changing requirements may slow development progress.

f. Tailoring.

(1) Another useful display for this metric is the effect of requirements changes on the code. This can be seen by plotting the percent, or number, of SLOC changed over time.

(2) Function points could be tracked rather than SLOC.

(3) Requirements stability metric tailoring should consider the criticality of individual requirements and units. For example, if user requirements are prioritized, tailoring might consist of tracking changes made only to software requirements that implement priority 1, 2, and 3 user requirements.

10–13. Design stability metric

a. Description. This metric is composed of two measures. The design stability measure tracks changes made to the design of the software. The design progress measure shows how the completeness of the design is advancing over time and provides a context for viewing the design stability measure in relation to the total projected design.

b. Application.

(1) *Data collection and reporting.* Begin tracking as code is entered into configuration management and continue for each version until completion. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* Specific preparation steps are not needed for this metric.

c. Data definitions. For each CSCI and each delivery/design version collect—

(1) Date planned for design/delivery version completion.

(2) M = Number of units in current delivery/design.

(3) F_c = Number of units in current delivery/design that include design related changes from previous delivery.

(4) F_a = Number of units in current delivery/design that are additions to previous delivery.

(5) F_d = Number of units in previous delivery/design that have been deleted.

(6) T = Total number of units projected for system.

d. Presentation and analysis.

(1) Plotting the calculated design stability (S) and design progress (DP) values over time as in figure 10–13 is a recommended display. Table 10–8 has the formulas for the two design measures.

Table 10–8
How to compute design stability measures

Formula	Where
$S = [M - (F_a + F_c + F_d)] / M$	S = design stability measure
DP = M/T	DP = design progress measure

(2) Although not indicated in figure 10–13, it is possible for design stability to be a negative value. This may indicate that everything previously delivered has been changed and more units have been added. If the current delivery contains fewer units than the previous one, a negative value indicates that the number of units deleted or changed from the previous baseline was greater than the total number of units in the current delivery.

(3) If some units in the current delivery are to be deleted from the final delivery, it is possible for design progress to be greater than one.

e. Management information.

(1) The design stability measure depicts how much of a software delivery, or version, is comprised of pieces reused without modification from the previous delivery or version. The closer this value is to one, the higher the amount of reuse.

(2) The design stability measure should be monitored to determine the number and potential impact of design changes, additions, and deletions on the software configuration. The trend of the measure over time indicates the software design is approaching a stable state when the curve levels off at a value approaching one. In addition to a high value and level curve, the following other characteristics of the software should be exhibited:

(a) The development progress metric is high.

(b) Requirements stability is high.

(c) Depth of testing is high.

(d) The fault profile curve has leveled off and most software problem/change reports have been closed.

(3) The higher the design stability measure, the better the chances of a stable software configuration. However, a value close to one is not necessarily good unless M is close to the total number of units required in the system (design progress measure approaching one), and the number of changes being counted are relatively small and diminishing over time. Periods of inactivity could be mistaken for stability.

(4) When design changes are being made to the software, the impact on previously completed testing must be assessed. Tests may need to be redone and may require modifications to test data and conditions.

(5) Allowance for exceptional behavior of this metric should be made for the use of rapid prototyping. Prototyping, while possibly causing lower design stability numbers early in the program, should reduce the number of design changes needed during later stages of development.

(6) The PM should establish criteria to define what constitutes a " design change." A design change implies change to the code for specific reasons, not a change due to style or coding preferences, or to add comments.

(7) Be aware that this metric does not measure the extent or number of changes in a software unit nor the quality of its code. Other metrics, such as complexity, can contribute to such an evaluation. This metric also does not identify the specific units that are being changed.

(8) The design stability metric can be used in conjunction with the complexity metric to highlight changes to the most complex units. It can also be used with the requirements metrics to highlight changes to units which support the most critical user requirements.

(9) If tracking design stability for builds or increments, T will likely be less than the total number of units projected for the system, but will reflect the total projected for the build.

(10) The relation of design stability with other metrics is shown in table 10–9.

f. Tailoring.

(1) Graphs of T and M over time are also useful. One would expect the projected number of units at completion (T) not to vary significantly from reporting period to reporting period with the exception of occasional replanning actions. Likewise, the number of units in each successive delivery would not be expected to fluctuate widely or to steadily decrease.

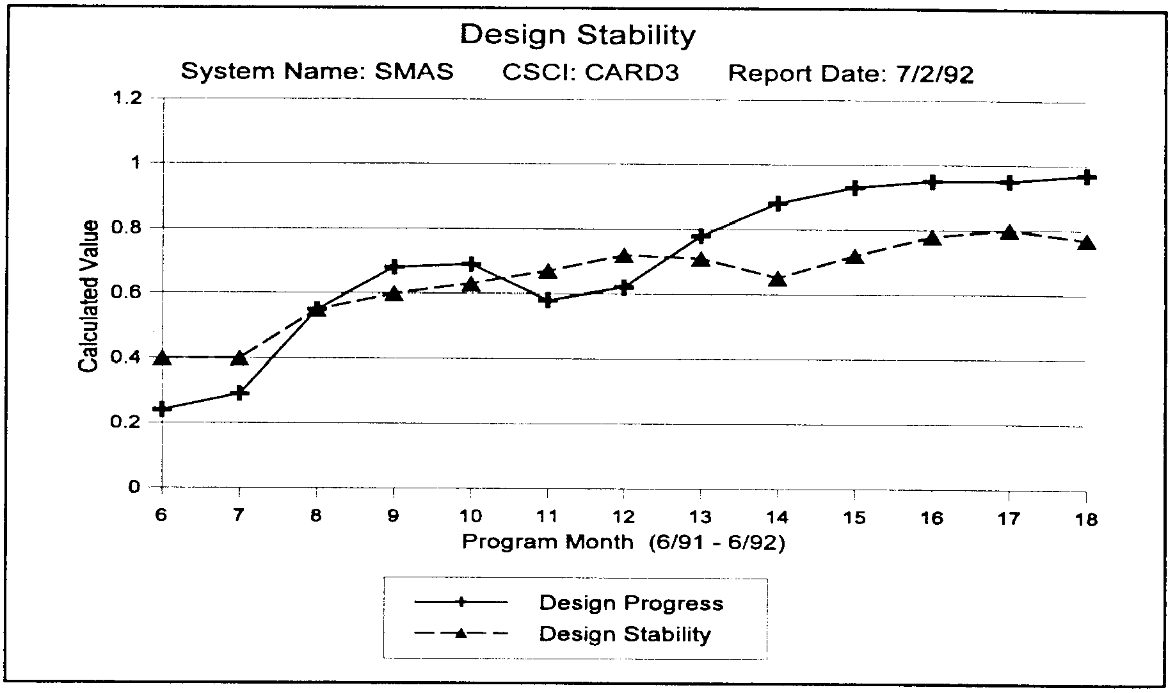


Figure 10-13. Sample design stability and design progress graph

(2) Defining any modification of code as a design change, regardless of reason, may be easier to implement than monitoring only design related changes, depending on the configuration management tools being used.

Table 10–9
Design stability relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by unusually large numbers of design changes.
CRU	What is the impact on resource utilization of adding/modifying software units? Are target upper bounds being approached or exceeded?
Requirements traceability	Everything implemented in the code should be traceable back to the documented design of a unit, CSCI, and higher level specifications. Also, have all requirements been implemented in the code?
Complexity	Compute the complexity values for new and modified units. A steady growth in the proportion of units with high complexity could indicate insufficient attention to applying good design and coding practices.
Breadth of testing	If changes are a result of “ bug” fixes run regression tests to verify changes.
Depth of testing	Obtain depth of testing data for new and modified units.
Development progress	Changing design may slow development progress.

10–14. Complexity metric

a. Description. The complexity metric provides a means to measure and evaluate the structure of software units. Software that is more complex is harder to understand, test adequately and maintain. Additionally, a highly complex unit is more likely to contain embedded errors than a unit of lower complexity. The likelihood of introducing errors when making code changes is higher in complex units. The Army complexity metric allows selection from five different measures, as shown in table 10–10. Each measure captures a different aspect of complexity.

b. Application.

(1) *Data collection and reporting.* Begin collecting McCabe’s cyclomatic complexity during software design. Begin collecting other complexity measures as units are placed under the developer’s configuration control. Recompute the complexity measures for units after they are modified during development and PDSS. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.*

(a) Specific preparation steps are not needed for this metric. Source code or program design language is the material examined for determining complexity. Using automated tools to compute the measures accurately and consistently is strongly recommended. A brief discussion of each complexity measure follows.

(b) McCabe’s cyclomatic complexity is based on graph theory. It is the number of independent control paths through a unit, from entry point to exit point (also called basis paths). The lower the number of independent paths means fewer tests are needed to exercise all possible control sequences in that piece of software. Cyclomatic complexity is calculated with the formula in table 10–11. An illustration of a flow graph and its cyclomatic complexity derivation is shown in figure 10–14.

(c) The three Halstead measures included in the complexity metric are expressions of program size based on the data manipulated in a program (operands) and the operations performed with the data (operators). The measures are calculated with the formulas in table

10–12. The larger a unit's size, more effort is needed to understand, test, and maintain it.

Table 10–10
Measures comprising the complexity metric

Measure	Quantifies the attribute(s) of
McCabe's cyclomatic complexity	Relative degree of effort to test or maintain a software unit (based on the number of ways control could flow through the unit)
Halstead's length, vocabulary and volume	Relative degree of effort to test or maintain a software unit (based on the amount of data and number of operations performed on them)
Control flow	" Unstructured" changes in control flow through the unit
Source lines of code (SLOC)	Size
Percent comment lines	Understandability and maintainability

Table 10–11
How to compute cyclomatic complexity

Formula	Where
$C = E - N + 2P$	C = the cyclomatic complexity E = # of edges (program flows between nodes; i.e., branches) N = # of nodes (groups of sequential program statements) P = # of connected components (number of disconnected parts on a flow graph)

Table 10–12
How to compute Halstead size measures

Formula	Where
$v = n_1 + n_2$	v = program vocabulary n_1 = # distinct operators n_2 = # distinct operands
$L = N_1 + N_2$	L = program length N_1 = total # of occurrences of the operators N_2 = total # occurrences of the operands
$V = L(\log_2 v)$	V = program volume

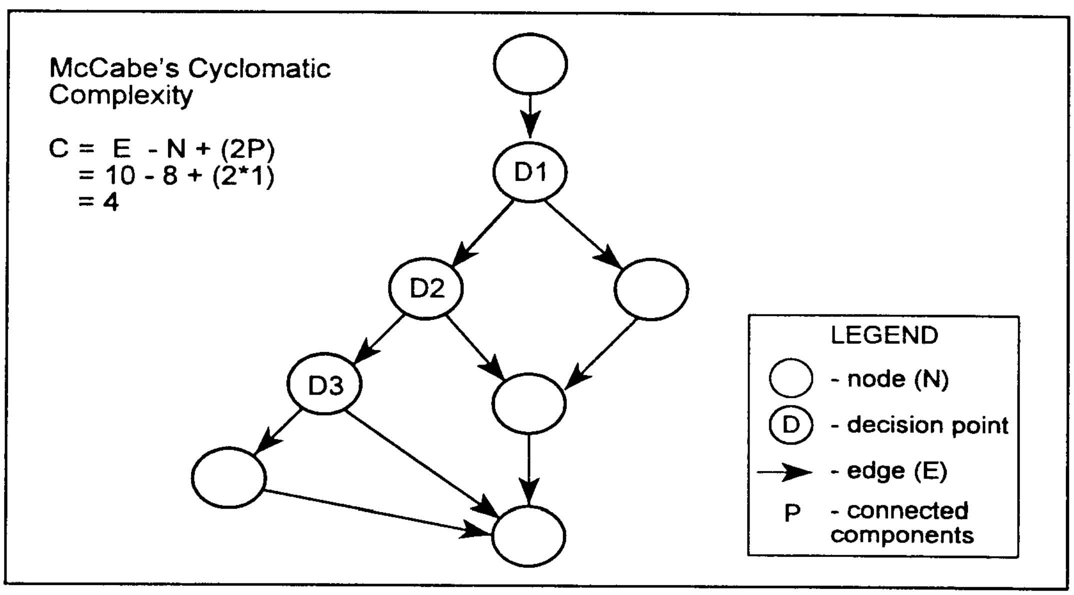


Figure 10-14. Example flow graph and cyclomatic complexity

(d) The control flow measure counts the number of times control paths cross in a unit. Control path crossings are also referred to as " knots." For example, a conditional GOTO statement embedded within a standard structured control flow construct (for example, WHILE, UNTIL, CASE) that sends control out of the construct improperly would cause a knot to occur in a unit's flow graph. The control flow value for figure 10–14 is zero.

(e) The SLOC size measure is the number of lines of code (LOC) in a unit. Lines of code are defined as non-comment, non-blank, executable, and data statements. Source LOC refers to code which is written and maintained by programmers.

(f) The percent comment lines measure shows the relative amount of explanatory material in a program compared to its size, in lines. For the purposes of computation, blank lines are not considered comment lines. The percent comment lines for a software unit is calculated as (C / T) * 100, where C = number of comment lines and T = total number of non-blank lines in the unit.

c. Data definitions. The data to collect for this metric depends on which of the five complexity measures are desired. The measures are calculated at the unit level, where a unit is defined as the smallest piece of testable code. For each software unit in each CSCI collect—

(1) Programming language the unit is written in.

(2) For the McCabe's cyclomatic complexity measure, the calculated value of cyclomatic complexity.

(3) For the Halstead measures, the calculated values of program vocabulary, program length, and program volume.

(4) For the control flow measure, the number of control path crossings.

(5) For the SLOC measure, the number of lines of code.

(6) For the percent comment lines measure, the calculated percentage of comment lines.

d. Presentation and analysis. Recommended displays for this metric allow comparing a program's complexity with threshold values generally accepted as distinguishing complex software from simpler software (see table 10–13). Figure 10–15 is a histogram of a CSCI's units distributed by cyclomatic complexity value.

Table 10–13
Thresholds to minimize complexity

Measure	Suggested threshold per unit
McCabe's cyclomatic complexity	≤ 10 for source code ≤ 7 for PDL
Halstead's volume	≤ 3200
Control flow	= 0
SLOC	≤ 200
Percent comment lines	≥ 60%

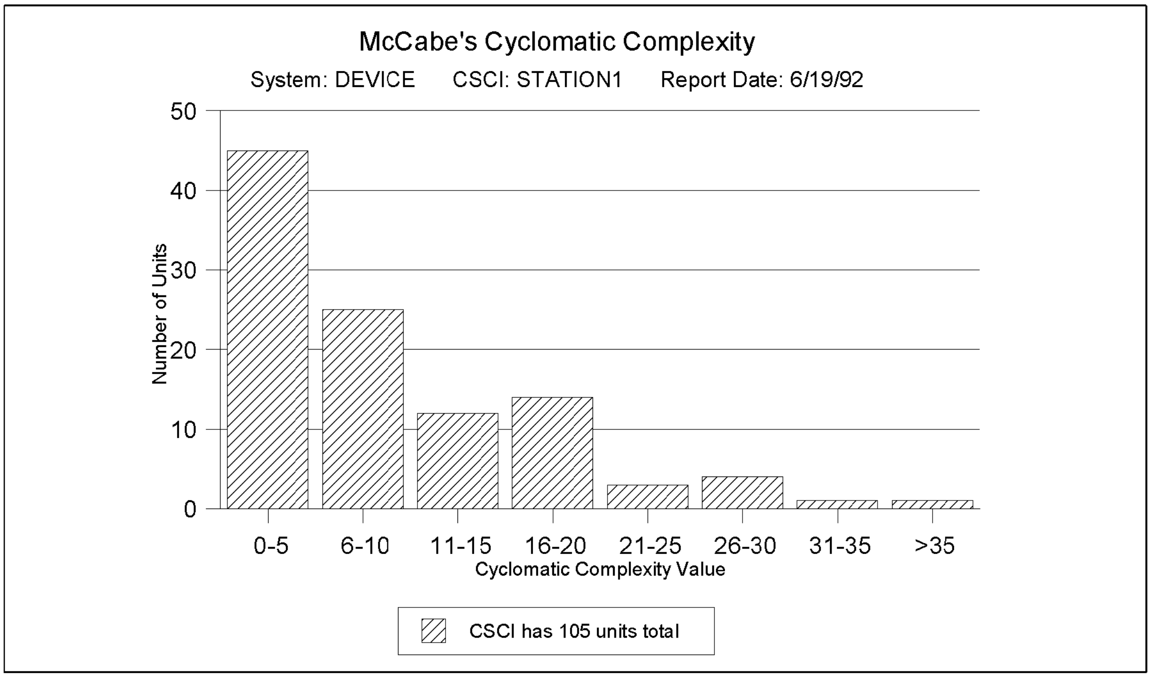


Figure 10-15. Sample cyclomatic complexity display

e. Management information.

(1) Automated tools are available for many programming languages and software development environments and should be used to assist in computing the complexity measures.

(2) This metric applies throughout the software life cycle. Establishing a complexity threshold during development stimulates structured programming techniques and limits the number of critical paths in a program during design and unit implementation. Complexity is used during software testing to identify basis paths, define and prioritize the testing effort, and assess the completeness of unit testing. During PDSS, proposed changes that would substantially increase complexity should be examined closely, as they could also increase testing effort and decrease maintainability.

(3) It is recommended that this metric be used as soon as it is practical. It is highly desirable to limit the inherent complexity of software during design and as code is being developed. Although the metric provides valuable information, it should not be relied upon as the sole metric to judge the quality of the design's implementation. Chapter 5 lists test and CE activities used in conjunction with design. Wherever possible, complexity should be computed for PDL during the design activity. The suggested cyclomatic complexity threshold for PDL is lower than the source code threshold to allow for expected growth during unit implementation (see table 10–13).

(4) Complexity measures should be generated for each unit in the system. They can be grouped for display in a number of ways (for example, by CSCI, by individual unit, and so forth). Examining complexity at various levels can provide indications of potential problem areas. These indications give guidance to the developer on areas where additional concentration is needed. The Government can use complexity to find areas where test efforts should focus, such as performing code walk-throughs, more comprehensive unit level testing, or stress testing. While the majority of units in figure 10–15 have values less than or equal to ten, it can be seen that several units have well exceeded this suggested threshold. These units should be examined closely through testing and analysis.

(5) There are several embedded assumptions and known weaknesses in the complexity measures. For example, in computing McCabe's cyclomatic complexity, there is no differentiation between different kinds of control flows. A CASE statement, which is

easier to use and understand than a corresponding series of conditional statements, makes a high contribution to cyclomatic complexity. This is counterintuitive considering that the corresponding series of IF...THEN...ELSE statements would be more trouble to test, modify, and maintain. An exception to the cyclomatic threshold in table 10–13 may be appropriate for large CASE statements where a number of independent blocks follow the selection function. Further, a million straight line instructions have the same cyclomatic complexity as a single instruction. Using more than one complexity measure, however, can offset the shortcomings of any single measure.

(6) There are many ways of defining and counting lines of code. The fairly simple definition (non-comment, non-blank, executable, and data statements) is intended to apply somewhat equally across the spectrum of procedural languages. The purpose of counting lines of code in this metric, as well as in other metrics, is to indicate the relative amounts of change in units as they are built and maintained, as well as to indicate unit size.

(7) The percentage of comment lines is a language-dependent measure. Some self-documenting languages require fewer comments than an assembly language or a language like FORTRAN. Additionally, the measure does not address the usefulness or completeness of the comments.

(8) The complexity measures outlined here are oriented towards procedural programming languages. They are appropriate for similar procedural environments and the large volume of legacy code currently being maintained and enhanced. When applied to artificial intelligence and pure object-oriented languages, however, use care when interpreting the results. Additional complexity or design structure measures for languages like Ada (to measure the degree of encapsulation, for example) can lend further insights into the software structure and their use is encouraged.

(9) In cases where units have a high cyclomatic complexity (many independent control paths), various techniques exist to help identify how complexity may be reduced. One method assesses the unit's actual complexity to identify control paths that cannot be tested. This can occur when a program's data flow and data conditions at various decision points preclude control from ever taking those paths. These sections are candidates for rewrite or elimination. Another method examines essential complexity, a gauge of the use of standard structured control constructs. These three types of complexity and guidance on designing a minimum set of control path tests are discussed in detail in National Bureau of Standards (NBS) 500–99.

(10) More than one complexity measure should be used because each assesses a different complexity attribute (see table 10–10).

(11) Units planned for reuse should not be overly complex.

(12) Examining complexity trends over time can provide additional useful insights, especially when combined with other metrics such as design stability or development progress. For example, late software code " patches" may cause the complexity of the patched unit to exceed an acceptable limit, indicating that the design rather than the code should have been changed. Test resources may be better expended on units that have a relatively high structural complexity rather than on units that will reflect a high number of lines of code tested.

(13) The relation of complexity with other metrics is shown in table 10–14.

Table 10–14
Complexity relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by exceedingly complex code.
CRU	If RAM target upper bounds are being approached or exceeded it may be necessary to optimize the code (generally making it more complex).
SEE	Organizations with a higher SEE rating generally develop less complex and easier to maintain code.
Depth of testing	Units with high complexity require more test resources.
Fault profiles	High complexity units may contain more faults.

f. Tailoring.

(1) Tailoring the amount of data collected for the complexity metric should consider criticality of individual units. For example, if user requirements are prioritized, tailoring might consist of tracking cyclomatic complexity only for software units that implement priority 1, 2, and 3 user requirements, while collecting SLOC for all units.

(2) SLOC has been retained as a measure of size and complexity because of its long history of use and relative ease of computation, especially for legacy systems. Other size measures, such as function points, may be used, but it is recommended that they supplement and not replace SLOC.

(3) The complexity measures in this metric do not address Ada issues, such as parallelism, data abstractions (tasks and packages), overloading, and generics. An Army manager may wish to collect and monitor information flow measures in addition to control flow measures.

(4) Cohesion and coupling are also valuable as complexity measures because they assess additional software design attributes such as the type of relationships that exist between logical elements in the same unit, and the relationships that exist between units, respectively. These measures are particularly helpful in non-procedural object-oriented implementations.

10–15. Breadth of testing metric

a. Description. Breadth of testing addresses the degree to which required functionality has been successfully demonstrated as well as the amount of testing that has been performed. This testing can be described as " black box" testing, since it is only concerned with obtaining correct outputs as a result of prescribed inputs.

b. Application.

(1) *Data collection and reporting.* Data collection should begin when any formal software testing is performed. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* Test cases must be developed to demonstrate specific requirements and assigned to test events, and test results assessed before data can be meaningfully gathered for this metric.

c. Data definitions.

(1) For each CSCI, each formal test, and each requirement type collect—

(a) Type of requirements tested and evaluated (such as SRS, IRS, UFD).

(b) Total number of that type of requirement allocated to the CSCI.

(c) Number of requirements tested with all planned test cases.

(d) Number of requirements successfully demonstrated.

(e) Test identification (for example, UAT, CSCI qualification testing, system qualification testing, DT, OT).

(f) Requirements priority or criticality, if any.

(2) The types of requirements in table 10–15 are recommended for tracking with this metric.

(3) It is advised to track software requirements (SRS, IRS) tested and passed through higher test levels beyond software qualification tests.

(4) This metric does not track the test progress of individual requirements. For that reason, it is advised that the " number of requirements" data items (para 10–15 *c*(1)(*a*)-(*c*)) be cumulative values across tests.

d. Presentation and analysis.

(1) The requirements counts collected in the breadth of test metric can be used to compute three different measures of testing progress: one is a measure of test coverage, and two are measures of test success as shown by table 10–16.

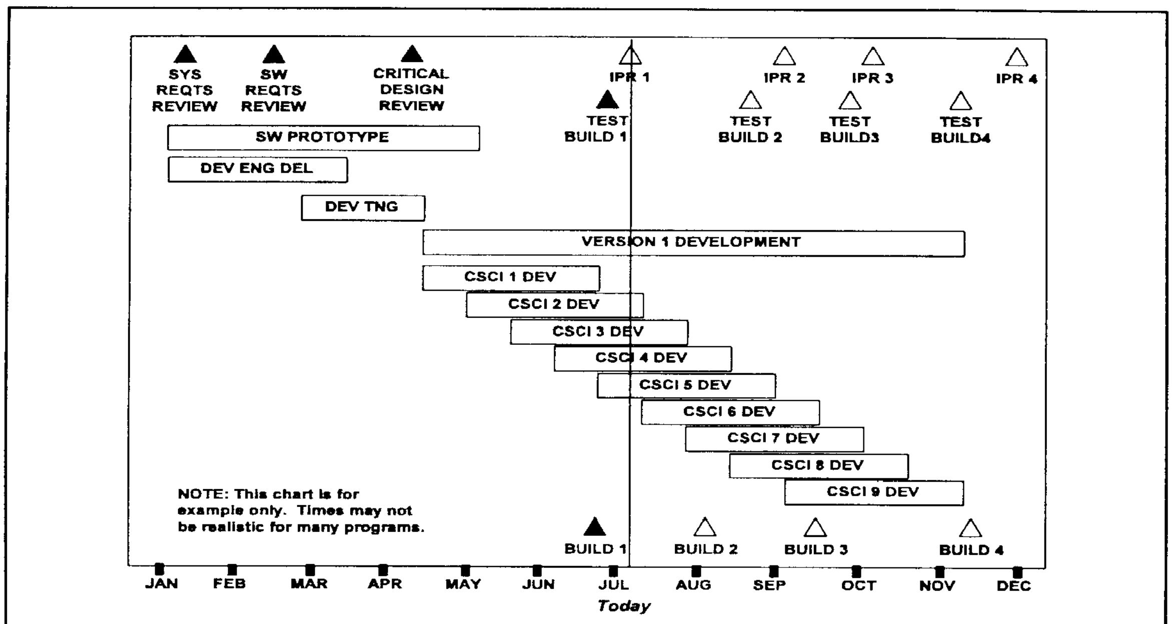
(2) All three test progress measures can be simultaneously displayed over key test events as shown in figure 10–16. It is recommended that test coverage and test success values be displayed as percentages by multiplying each value by 100.

Table 10–15
Recommended items for breadth of testing metric tracking

This type of requirement	Typically elaborated/implemented in	Prioritize by criticality
User requirements	Users' Functional Description (UFD), Operational Concept Description (OCD)	Yes
System requirements	System/Subsystem Specification (SSS), Interface Requirements Specification (IRS)	Optional
Software requirements	Software Requirements Specification (SRS), IRS	Optional

Table 10–16
How to compute testing progress measures

Measure	Formula	Where	Addresses
Test coverage	R_{tested}/R_{total}	R_{tested} = # of requirements tested R_{total} = total # of requirements	How much of total was tested, without regard to test success (extent of testing).
Test success	R_{passed}/R_{tested}	R_{passed} = # of requirements passed	How much of what was tested was successful.
Overall success	R_{passed}/R_{total}		How much of the total was tested and successful.



This schedule shows current status in July. One cannot see how it compares to the original program schedule.

Figure 10-16. Sample testing progress graph

(3) Be aware that the test success measure is computed for a different population of requirements than the other two measures, test coverage and overall success. Therefore, test success may be higher than test coverage when expressed in percent.

(4) It is possible, in some cases, that test coverage may never reach 100 percent. This can occur if all documented requirements are not demonstrated through tests, but are qualified by some other means, such as inspection of source code for adequate commentary for maintenance.

e. Management information.

(1) The breadth of testing metric measures the quantity of testing performed and achieved on documented requirements. While most requirements are usually functional, the metric also captures the results of performance, recovery, safety, security, adaptation, and any other requirements imposed by the acquirer that can be demonstrated through testing.

(2) The overall success measure provides insight into the level of progress made toward implementing the approved requirements baseline.

(3) Any change in the software requirements baseline requires recalculating the breadth of testing measures.

(4) Data should be collected throughout developmental test activities, if possible. Typically, breadth of testing is collected for CSCI qualification testing and system-level tests.

(5) The breadth of testing metric should also be reported incorporating the results of Government tests, such as DT and OT, particularly if there are requirements that cannot be adequately demonstrated prior to these system tests.

(6) The PMs should be aware of which software requirements cannot be tested until late in the testing process, or if a software function cannot be demonstrated at all prior to deployment.

(7) An innovative aspect of the Army's UFD is the option to assign a priority level to each user requirement to identify the most important requirements to be implemented in the software. The OCD, SSS, SRS, and IRS outlined in the consolidated software standards of paragraph 2–2 *d* also discuss provisions for assigning precedence or criticality values to requirements. Data for this metric may be collected and reported separately for each requirements priority level to provide more detailed visibility into which requirements are being tested.

(8) As requirements are added and deleted over time, the population of total requirements also changes. This can cause the reported breadth of testing measures to fluctuate for reporting periods when no testing was performed.

(9) When changes are made to requirements or design, previous test results for those areas are no longer valid. Until retesting and re-evaluation of results occurs, the number of requirements tested and number of requirements passed reported in breadth of testing should drop by the number of requirements to be retested.

(10) Without clear criteria for test success, the breadth of testing metric may not be effective, due to the subjectivity in assessing whether a requirement has actually been satisfied.

(11) The relation of breadth of testing with other metrics is shown in table 10–17.

f. Tailoring. Depending on how test success criteria are established, failing only one test case may result in a requirement not being successfully demonstrated. If sufficient resources exist, an optional way to display breadth of testing is to report the percentage of test cases performed and passed for each individual software requirement. This procedure gives managers insight into the amount of testing done on each requirement (assuming multiple test cases exist for a requirement). This optional method of data collection and reporting may be useful for especially critical requirements, but impractical for all requirements due to cost considerations.

10–16. Depth of testing metric

a. Description. The depth of testing metric measures the amount of testing achieved on the software architecture. That is, the extent and success of testing the possible control and data paths and conditions within the software. This testing is often described as" white box" testing, since there is visibility into how the software is constructed.

b. Application.

(1) *Data collection and reporting.* Begin collecting data at the time that a configuration controlled code baseline is available for unit testing. This metric should also be reported to reflect regression testing as changes occur in the baseline during development and PDSS. The recommended reporting frequency for this metric is monthly.

Table 10–17
Breadth of testing relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by an unusually large number of problems uncovered in testing.
Requirements traceability	During PDSS, the breadth of testing metric should be used with the requirements traceability metric to measure the level of regression testing needed.
Requirements stability	Changes in requirements require test cases to be modified or developed. Previous test results for changed requirements are no longer valid; tests should be rerun and breadth of testing recalculated.
Design stability	Changes in design driven by derived requirements require test cases to be modified or developed. Previous test results for areas changed are no longer valid; tests should be rerun and breadth of testing recalculated.
Complexity	CSCIs and systems comprised of many units with high cyclomatic or Halstead volume complexities will likely take longer to pass all their allocated requirements due to the greater number of test cases needed to adequately demonstrate the requirements.
Fault profiles	As faults are closed, subsequent retesting should show breadth of testing improving in test success.
Reliability	Low values for breadth of testing at fielding increase the risk of software reliability problems.

(2) *Preparation.* Four different attributes of software structure can be monitored by this metric. Selection of which of the four attributes to track, development of test cases to demonstrate them, assignment of specific tests to test events, and evaluation of test results need to occur before data can be meaningfully gathered for this metric. The four depth attributes and criteria for success are listed in table 10–18.

(3) As defined in the complexity metric, a path is a logical traversal of a unit from an entry point to an exit point, following a combination of edges and nodes. An edge is a program control flow between nodes. Nodes are groups of sequential program statements.

(4) The statement attribute pertains to executable statements only.

(5) Each decision point which contains an " or" statement should be tested at least once for each of the condition's logical predicates.

c. Data definitions.

(1) For each unit in each CSCI, and each software depth attribute monitored, collect—

(a) Name of the depth attribute.

(b) Total number of attribute occurrences.

(c) Number of occurrences executed at least once.

(d) Number of occurrences successfully executed at least once.

(2) It is recommended to track at least paths, statements and inputs with this metric.

d. Presentation and analysis.

(1) The attribute counts collected in the depth of testing metric can be used to compute two measures of testing progress: one is a measure of test coverage, and the other is a measure of test success as shown by table 10–19.

(2) The recommended display for this metric is a plot of test coverage and overall success for a depth attribute over time, expressed in percent, as depicted in figure 10–17.

Table 10–18
Software structure attributes measured by the depth of testing metric

Attribute	Success criteria	Recommended for routine tracking
Path	Path is successfully executed at least once	Yes
Statement	Statement is successfully executed at least once	Yes
Input	Input is successfully tested with at least one legal entry and one illegal entry [1, 2]	Yes
Decision point	Decision point is successfully exercised with all classes of legal conditions as well as one illegal condition [2]	Optional

Notes:
[1] An entry should be selected from every field of every input parameter.
[2] Successful test of an illegal entry or illegal condition means unplanned or undesirable results do not occur.

Table 10–19
How to compute test progress measures for depth attributes

Measure	Formula
Test coverage	$\frac{\text{Number of attribute occurrences tested}}{\text{Total number of occurrences of the attribute}}$
Overall success [1]	$\frac{\text{Number of attribute occurrences passed}}{\text{Total number of occurrences of the attribute}}$

Notes:
[1] Overall success is also termed path measure, statement measure, domain measure and decision point measure for path, statement, input, and decision attributes, respectively.

e. Management information.

(1) The depth of testing metric provides information on the integrity of the software design, including the relationship between the paths, statements, inputs, and decision points of the software.

(2) The depth measures discussed here do not assess the“ correctness” of design or code. It is expected that unit tests and unit integration and testing will make use of test cases that demonstrate code is designed properly. These cases should be supplemented by other cases to yield coverage and success measures that provide satisfactory confidence that unexpected control or data conditions will not occur. Software test programs usually require that software structure is successfully demonstrated only after passing some “ realistic” number of test cases, under both representative and maximum stress loads. It is understood that fully exhaustive testing of all control and data combinations is prohibitive.

(3) Because illegal inputs are used, the domain measure provides an indication of the robustness of the software design.

(4) Some judgment is required to interpret the domain measure because it is unlikely that the program will be subjected to all possible input streams. However, the domain measure is important because most faults appear at domain boundaries.

(5) The relation of depth of testing with other metrics is shown in table 10–20.

f. Tailoring.

(1) The recommended data definitions for this metric are collected for each unit. However, data may also be collected at the CSCI or system level if adequate test tools are available.

(2) Depth of testing data collection should be tailored to consider the effort to collect data. These guidelines are suggested:

(a) Always compute the domain measure (inputs).

(b) Always compute the path and statement measures over the set of basis paths (see complexity metric, para 10–14), on units that implement high priority requirements, or if a unit’s complexity values exceed established thresholds.

(c) Compute more comprehensive path and statement measures if automated tools are available.

Table 10–20
Depth of testing relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by unusually large numbers of problems uncovered in unit test.
SEE	Organizations with maturity rating of 1 may not have well-established procedures for ’white-box’ testing.
Requirements stability, Design stability	Changes in requirements or design may lead to modifying code or developing new code. Previous test results for changes are no longer valid; tests should be rerun and depth of testing recalculated.
Complexity	Structural testing of a unit with high cyclomatic or Halstead volume complexity will take longer than one of lesser complexity due to the greater number of control and data paths through the unit. Units of highest complexity should be designated for early and more thorough testing.
Breadth of testing	Units with low depth coverage measures may cause breadth measures to be low as well.
Fault profiles	Low values of overall success should correlate with software errors reported in the fault profiles metric.
Reliability	Low values for the depth of testing metric increase the risk of software reliability problems.

10–17. Fault profiles metric

a. Description. The fault profiles metric is a summary of software problem/change report (PCR) data collected by the corrective action system as described in chapter 8. This metric provides insight into the number and type of deficiencies in the current software baseline, as well as the developer’s ability to fix known faults.

b. Application.

(1) *Data collection and reporting.* Collection begins early in the software life cycle when the first software product, usually a requirements definition document, has been approved and placed under configuration control. Continue to collect fault profiles data for the life of the program. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* A corrective action system is the source of problem/change report, or fault, information for this metric. Chapter 2 provides a method for uniformly prioritizing and categorizing problems and changes. In order to compute the age of faults, individual faults need to be tracked by the corrective action system, with the dates of problem start and problem closure recorded.

c. Data definitions.

(1) For each CSCI, each fault priority, and each fault category, collect—

(a) Cumulative number of faults detected.

(b) Cumulative number of faults closed.

(c) Average age of open faults.

(d) Average age of closed faults, which is the same as average time to close.

(e) Average age of all faults.

(2) Average ages can be computed using the formulas in table 10–21.

d. Presentation and analysis. The displays discussed here can be organized by any desired grouping of fault priorities and fault categories.

(1) *Fault history.*

(a) History to date. A common display of fault profiles metric data is shown in figure 10–18 as the cumulative numbers of software faults detected (problem reports opened) and closed, over time. Cumulative fault history displays often show large differences between reporting periods, appearing as “ steps” in the curves. A large number of faults may be opened in a particular month, due to faults observed during a formal review, software audit, or test. Conversely, a cleanup period prior to a major test or software fielding release may show a large number of problems closed.

(b) Detailed detection/resolution history. An alternate display of corrective action activity is to plot the number of problem/change reports that were opened and the number closed over periodic intervals, such as months. Figure 10–19 is an example of monthly PCR opening and closure activity for one CSCI.

(2) *Average age of software faults.* Average fault age can be plotted over time to expose trends in fault resolution. Fault age graphs can indicate which CSCIs and which problem priorities are the most troublesome with respect to fixing faults. Figure 10–20 is a graph of the average length of time a fault not yet resolved has been in the corrective action system.

Table 10–21
How to compute average fault ages

Average age of	Formula	Where	
Open faults only	$D_{open}/Faults_{open}$	D_{open} =	the sum of the days between the time each open fault was detected and the current date
		$Faults_{open}$ =	total # of open faults
Closed faults only	$D_{closed}/Faults_{closed}$	D_{closed} =	total # of days all closed faults remained open
		$Faults_{closed}$ =	total # of closed faults
All faults (open and closed)	$D_{open}/Faults_{total}$	$Faults_{total}$	total # of open and closed faults

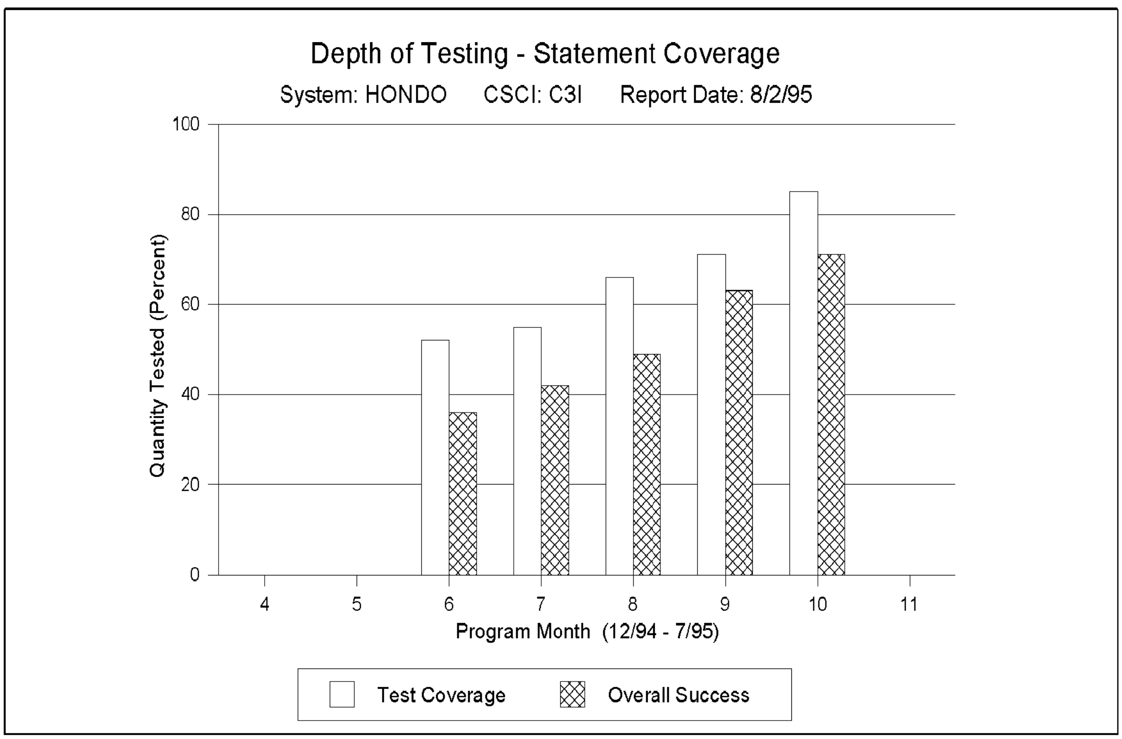


Figure 10-17. Sample depth of testing graph of statement measure

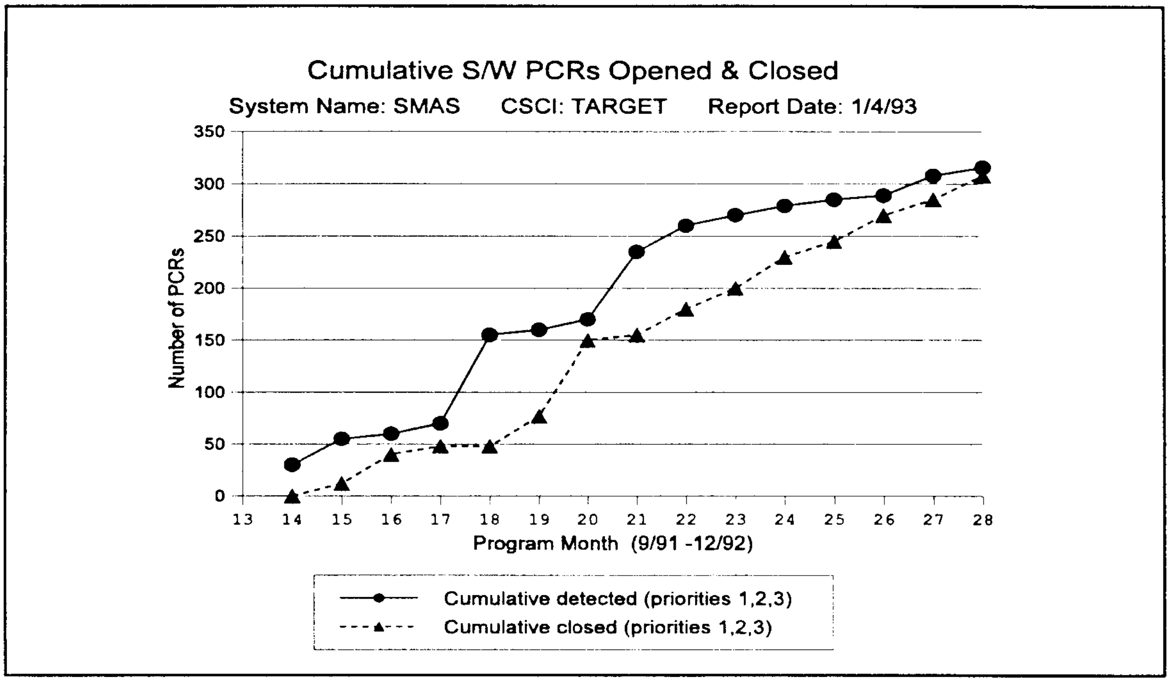


Figure 10-18. Sample graph of software problem history

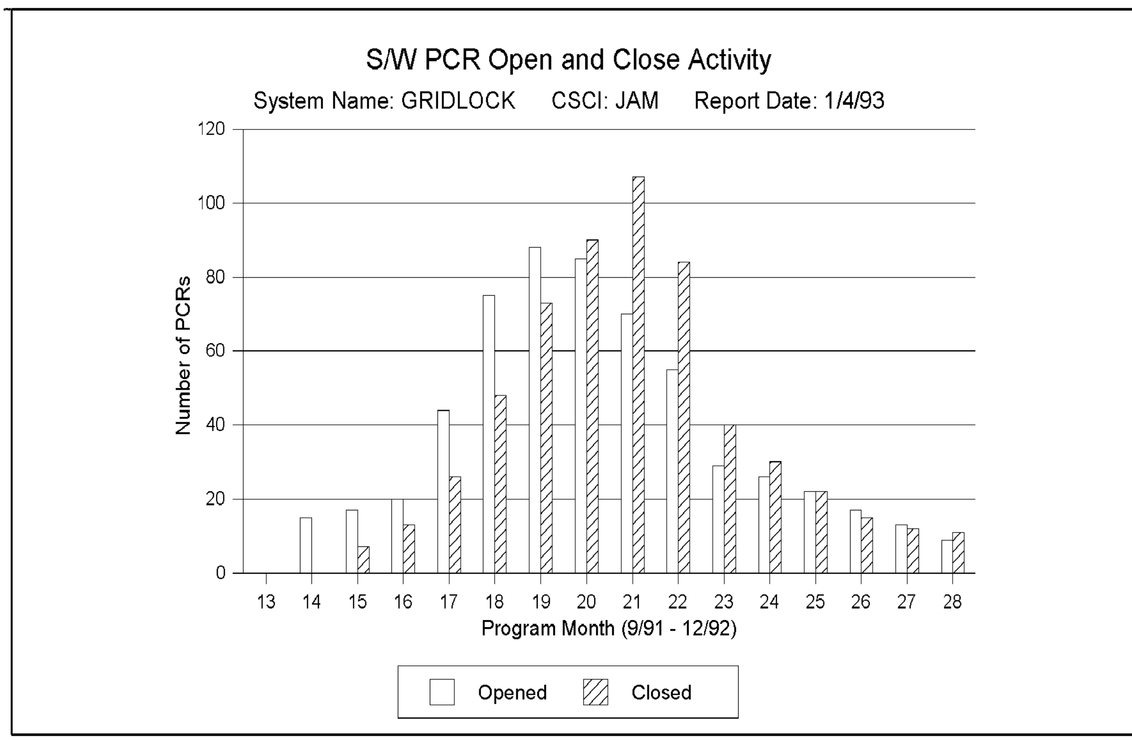


Figure 10-19. Example of monthly PCR activity

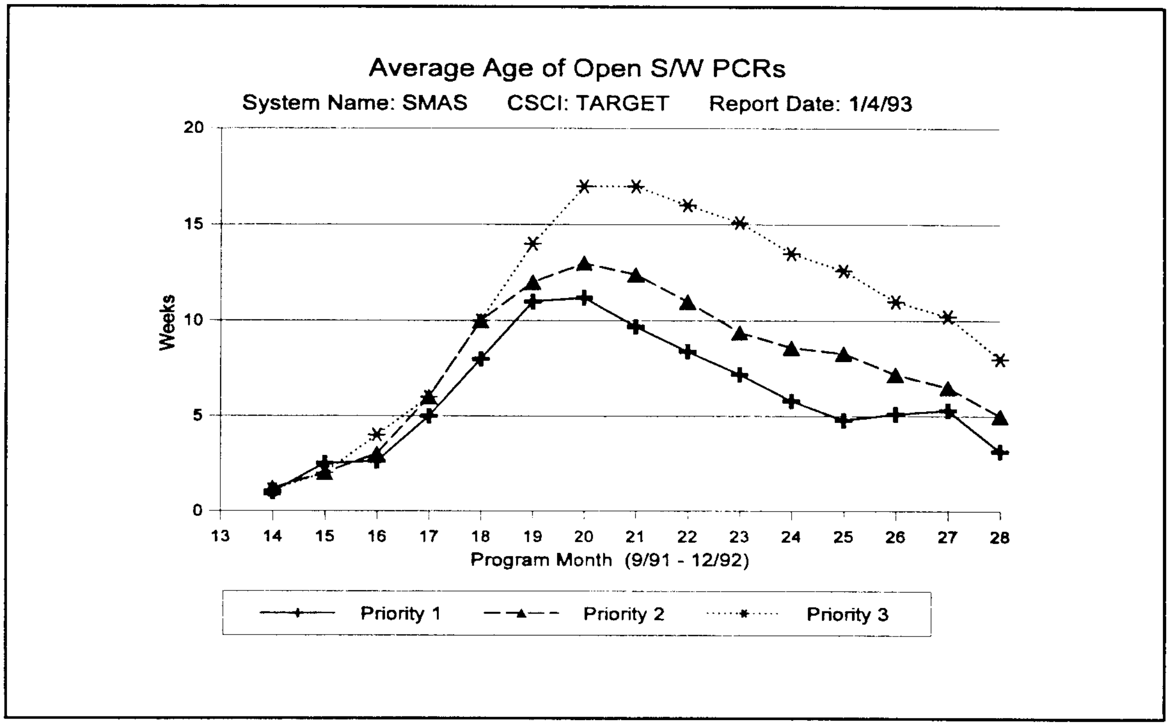


Figure 10-20. Sample graph of average age of open faults

e. Management information.

(1) Fault counts should be based on all tests and evaluations on a formal baseline which is under configuration control. Results of informal test-fix-test performed at the unit level should not be counted.

(2) The gap between open and closed faults should be closely monitored. A constant gap or a continuing divergence is reason for the user representative to take appropriate action, especially when approaching a key test or milestone.

(3) Inadequate problem resolution by the developer can cause the cumulative number of closed faults to remain constant over time, and a number of faults will remain open. The age of the open faults should be checked to see if they have been open for an unreasonable period of time. Those faults which are not resolved represent an increased risk. Managers should identify the reason that faults are not closed and take corrective action.

(4) Managers should be aware of the cumulative effect of a large number of low priority faults. Too many minor problems may impair overall system operation or successful test conduct. PMs may wish to establish thresholds to limit the cumulative effects of unresolved priority three and lower faults on cost or ability to operate the system effectively

(5) Army policy for the acceptable numbers of open software faults prior to certification of readiness for dedicated operational testing is outlined in section VIII of chapter 6. At a minimum, all priority one and two faults must be closed.

(6) The PM should establish a clear description of when a fault is considered discovered and closed. Criteria for the date discovered may be the date on which the original problem report was written, or when the report was entered into the corrective action system. Criteria for the date closed should reflect the CCB's judgement that regression testing was adequate and applicable documentation is updated. Differences in defining corrective action event dates can significantly influence the average ages reported via this metric.

(7) Average age graphs can track whether the time to close faults is increasing over time. Increasing time to close faults may indicate that the developer is not allocating adequate resources to correcting problems, or that some faults are exceedingly difficult to fix.

(8) Large deviations of individual faults from the average age of all faults should be investigated. The average open age of high-priority faults should also be examined with respect to the time remaining to the next major test or milestone.

(9) Examining the categories of software faults can provide insight into the underlying problems. During the early stages of software development, the fault profiles metric reports the quality of translating software requirements into the design. Design faults suggest that requirements were not defined correctly, or that they are being misunderstood by the developer. Later, the fault profiles metric measures the implementation of requirements and design into code, assuming an adequate level of testing is performed. Code faults could result from an inadequate design, or a poor job of implementing the design into code. Examining the fault categories to determine causal relationships should be performed in any analysis of fault profiles. Be aware that a single fault may be assigned to one or more categories.

(10) The PM should understand any fault or " bug" tracking tools used by the developer for tracking fault profiles data. The developer's system for collecting problem reports should be reviewed early in the program to determine how much of a difference there is between the recommended data definitions above and the definitions used by the tool.

(11) The PM should establish criteria to determine when a fix must be validated and by whom (Government or developer SQA).

(12) The PM should examine the following issues which are not reported in the fault profiles data:

(a) Time/cost of correction. The cost and time to correct a fault is not directly linked with the fault's priority. Priority one faults may be caused by trivial errors in syntax, while priority four faults may require a redesign.

(b) Problem description/prioritization is not always obvious. For example, a single character error in a source statement which leads to an improperly executed function. Interpretations of problem and priority may be different depending on whether the cause or effect

is emphasized. Priorities in table 2–4 are fairly straightforward. The method for determining fault categories and defining fault priorities is not as important as applying the definitions consistently.

(c) Category of fault. Faults in requirements are often the most expensive and persist the longest. These faults may not be detected until the software is used on site. Design faults could be related to processing or control flow. If these faults persist past unit-level testing, check inputs tested as reported in the breadth of testing metric. Control and sequence faults in code may include missing paths, unreadable code, loop termination criteria incorrect, uncontrolled GOTOs, spaghetti code (old COBOL). These faults are often caught with path testing. If many of these types of faults persist past unit testing, check the depth of testing metric for completeness.

(13) The fault profiles displays do not identify which individual faults persist over time. The developer's corrective action system may identify the software unit related to a fault to indicate product status. With unit identifiers, it may be possible to identify problem units and combine analysis with other metrics for a more complete diagnosis.

(14) When interpreting fault profiles data be aware that error detection is closely tied to the quality of the development and testing process. A low number of detected faults could indicate either good process management with good products, or a process with an inadequate amount or improper type of testing. Fault profiles metric data should not be evaluated without also considering measures of test coverage. For example, a plot of code category faults could be evaluated against the amount of testing which was done in each month. The relationship of code faults to test coverage can be used to gauge the maturity of software and the adequacy of the test program.

(15) Reliability models can be used to forecast the rate additional faults will be discovered based on previous error detection history.

(16) The relation of fault profiles with other metrics is shown in table 10–22.

f. Tailoring.

(1) The PM may want to track fault profiles more frequently than monthly during periods of heavy testing.

(2) It may be desirable to collect and report fault profiles data at the unit level for particularly complex, critical, or error-prone items.

(3) The test in which a fault was discovered can supplement the data for fault profiles metric (internal vs. Government, formal vs. informal).

10–18. Reliability metric

a. Description. The reliability metric assesses two aspects of software's ability to perform as intended. One set of measures expresses software's contribution to system mission reliability. System failures caused by software and the time it takes to restore the system to its previous operating condition after these failures occur are tracked. The other set of measures track summary data obtained from analytic models of reliability. Using data from the fault profiles metric, corrective action system, and test history, reliability models can project future failures as a function of test time (such as time to next failure or failure rate) and to project the number of latent, or as yet unobserved, faults remaining in a software baseline. These projections can be used to gauge how much testing will be enough to have confidence that critical faults will be within acceptable limits when the software is fielded.

b. Application.

(1) *Data collection and reporting.*

(a) Begin collecting the measures dealing with system failures caused by software during formal system-level tests, and continue through PDSS. The recommended reporting frequency for this information is monthly prior to deployment and as needed after fielding to reflect reported system failures. The system must be used under typical operating conditions for reliability data to be meaningful.

(b) Begin collecting software reliability model data during unit testing. The recommended reporting frequency for this information is monthly.

(2) *Preparation.*

(a) Software contribution to system mission reliability. Using fault profiles metric data, TIRs, and failure definition/scoring criteria (FD/SC) from the RRR, software analysts, and system RAM analysts need to mutually define and compute the items in table 10–23. Derive these items only when the software is used in accordance with its OMS/MP. Definitions and algorithms can be found in DA Pam 73–8.

Table 10–22
Fault profiles relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by unusually large numbers of software problems/changes. The later in the life cycle serious problems occur, the greater the affect's severity.
CRU	Problem reports may indicate computer resource capacity problems.
SEE	Developers with high SEE ratings should be expected to have superior problem reporting systems and low average ages of open faults.
Requirements traceability	Do test cases exist for each requirement that has been modified to correct a fault?
Requirements stability	A high number of requirements faults should generate a high level of requirements changes as the faults are fixed.
Design stability	Additional testing must be performed on new and changed units, possibly increasing the number of reported faults.
Complexity	High-complexity units often contain more faults. Ensure the test coverage for these units is high.
Breadth of testing	The number of faults discovered should be compared to the number of software requirements tested. Many faults with low coverage is undesirable.
Depth of testing	The number of faults discovered should be compared to the percentage of inputs, paths, and statements that have been tested. Many faults with low coverage is undesirable.
Reliability	Unresolved or unobserved faults in the software baseline may cause system reliability problems. Reliability models can be used to forecast the rate additional faults will be discovered.

Table 10–23
Computed items for software/system reliability tracking

Item	Description
1.	The point estimate of mean time between mission failures caused by system hardware or software as measured during the test event.
2.	The 80 percent lower confidence bound value of mean time between mission failures caused by system hardware or software.
3.	The point estimate of mean time between mission failures caused by software as measured during the test event.
4.	The 80 percent lower confidence bound value of mean time between mission failures caused by software.
5.	The mean time to restore the system to the operational condition existing before the failure, after a software-caused system failure has occurred.
6.	The median time to restore the system to the operational condition existing before the failure, after a software-caused system failure has occurred.
7.	The maximum 95th percentile value of time to restore the system to the operational condition existing before the failure, after a software-caused system failure has occurred.

(b) Software reliability modeling. In order to derive a predicted software failure rate and an estimate of latent software faults requires selecting an appropriate reliability model. Selecting a model is based on: the length of the test, how a software failure is defined, the adequacy of the operational profile, as well as the assumptions

underlying the use of each model. After choosing an analytical model, it is important to determine how closely the past predictions from that particular model for a particular data set reflect the actual behavior observed for that data set. Various statistical and qualitative methods can be employed to determine the degree of commonality between the two data sets. If the selected model is not an accurate reflection of the actual behavior then apply other models until a good fit is achieved. The most comprehensive collection of reliability prediction models is available in the Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) software package developed by the Government (see NSWCDD 84–373). Other software tools are also provided within SMERFS to compare a model's prediction of the behavior of a software application with the actual behavior of that software. Software faults used as input to the analytical models are described in the fault profiles metric, paragraph 10–17.

c. Data definitions.

(1) *Software contribution to system mission reliability.* For each system-level test event collect—

(a) Test identification.

(b) The required value of system Mean Time Between Mission Failure (MTBF).

(c) The required values for mean, median, and maximum 95th percentile mean time to restore the system to operational status.

(d) The items in table 10–23.

(2) *Software reliability modeling.* For each reporting period and each reliability model used, collect—

(a) Test identification.

(b) Name of the reliability model used.

(c) The software failure rate objective, such as desired goal for an acceptable number of software failures per month.

(d) The measured (actual) failure rate computed over the test period.

(e) The projected failure rate determined by the reliability model for the test period.

d. Presentation and analysis.

(1) Figure 10–21 shows computed point estimates of system mean time between mission failures, the required MTBF and associated 80 percent lower confidence bound plotted over time. A recommended display for mean time to restore system to operational status, also plotted over time, is shown in figure 10–22.

(2) Figure 10–23 is a display of software reliability model projection showing a steadily decreasing rate of software failures.

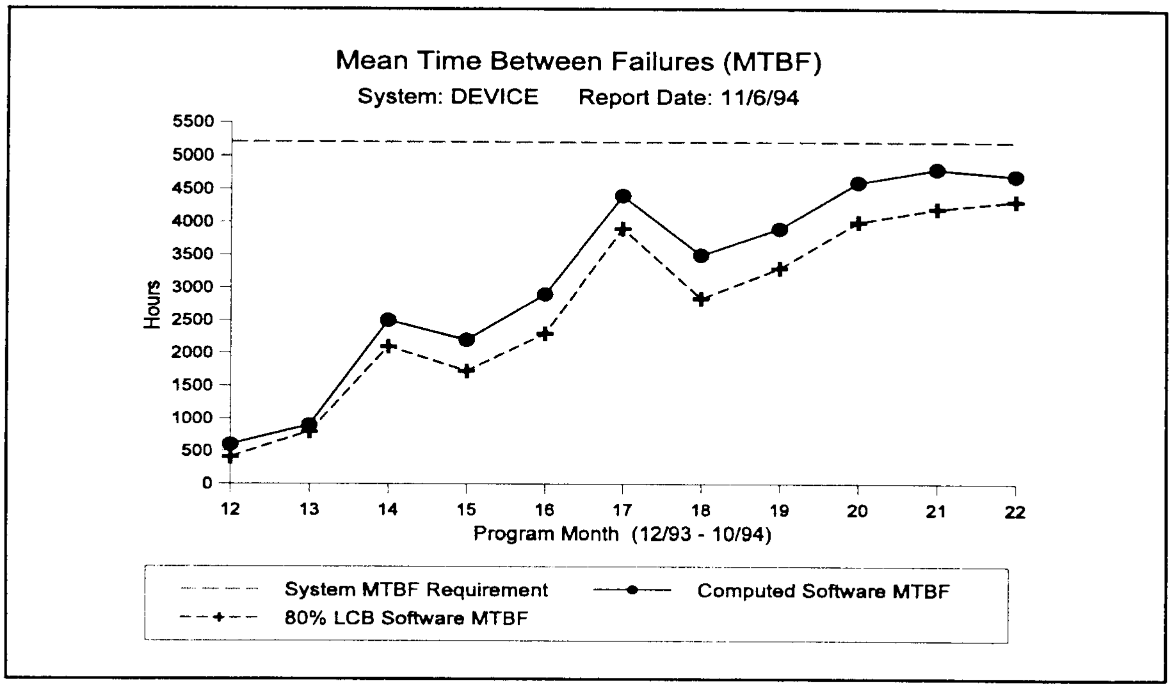


Figure 10-21. Sample graph of system mean time between mission failures

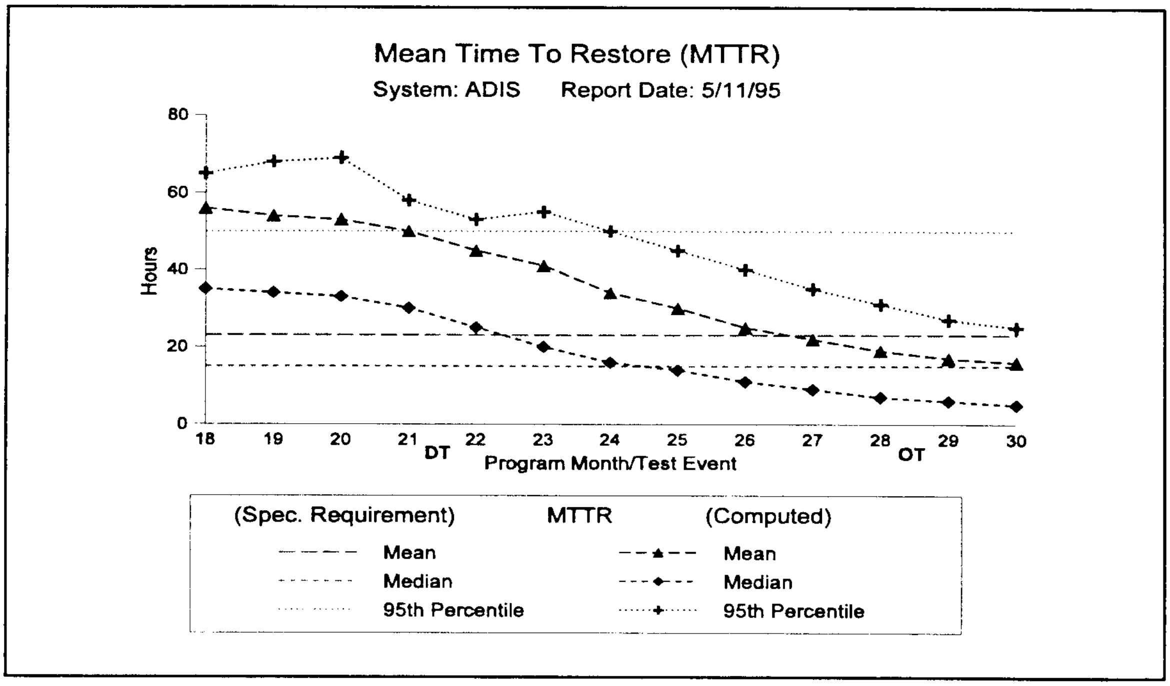


Figure 10-22. Sample graph of mean time to restore system

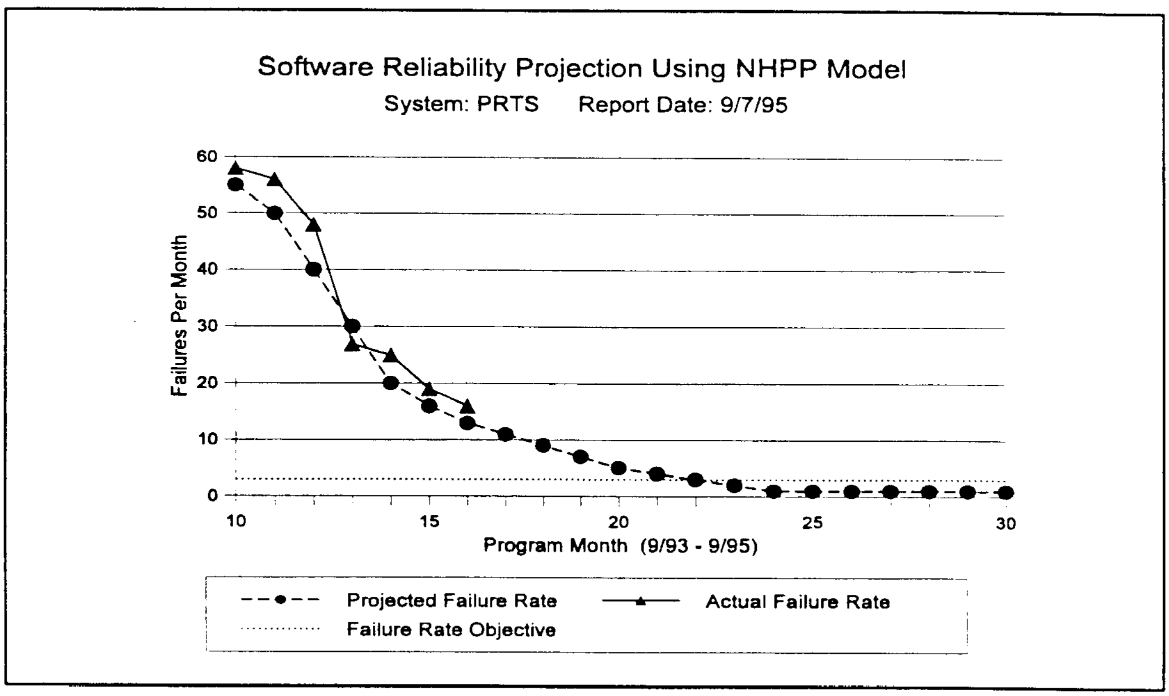


Figure 10-23. Sample graph of reliability model projection

e. Management information.

(1) The following information applies to calculating the contribution of software to system mission reliability.

(a) The user representative defines when the system is down in the system's FD/SC. Often, the system is characterized as down when it cannot perform specific critical functions or when a percentage of workstations or printers in the work area are not functioning.

(b) The system must be used under typical operating conditions for observed system mission reliability data to be meaningful. The system's operating conditions are documented in the OMS/MP.

When measured under these conditions, the reliability metric estimates how often one can expect the software to cause system failures in a field environment.

(c) Check reliability point estimates as each failure is encountered in DT to determine if the trend is moving towards the system's required reliability value. The PM should consider delaying OT until the computed MTBF is greater than the lower 80 percent confidence bound.

(2) Use the fault profiles metric to compare trends in the types of faults occurring and their rates of closure to projections software error populations and/or expected software failure rates.

(3) The PM should use reliability projections to gauge how much testing will be enough. These projections can be based on either system failures due to software or fault profiles.

(4) The relation of reliability with other metrics is shown in table 10–24.

Table 10–24
Reliability relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by unusually large numbers of system failures. The later in the life cycle a failure occurs, the greater the failure's severity.
CRU	Capacity shortages may lead to system failures.
SEE	A developer with a higher SEE rating is more likely to do reliability modeling.
Requirements traceability	Many software 'failures' are actually the result of system or user requirements which were not implemented in the final software code. Verify that traces are complete from requirements to code.
Breadth of testing	The reliability metric can be used to estimate the number of residual errors in the code and amount of additional system-level testing required.
Fault profiles	The data needed to compute fault profiles can be used for reliability measures.

f. Tailoring.

(1) Some user representatives would rather specify the operational availability (A_o) for their system, instead of MTBF. A_o is the percentage of time the system is either operating or is capable of operating. The data definitions for MTBF can be adjusted to reflect A_o.

(2) When tailoring, consider the criticality of the software in the system. Tracking MTBF is more appropriate for safety-critical systems. A_o is more appropriate for systems which can be safely shut down to resolve intermittent problems.

(3) Creating realistic operational conditions during pre-deployment testing can be expensive. Tracking reliability is easier during PDSS.

(4) When tailoring, consider the level of data necessary for each computed reliability item (see table 10-23).

(5) Collecting and reporting the estimated number of residual, or latent, faults may be desirable for systems that are highly critical or will be deployed for many years.

10–19. Manpower metric

a. Description. This metric provides an indication of the developer's application of human resources to the development program and ability to maintain sufficient staffing to complete the project. The manpower metric is composed of two parts: an effort measure monitors labor hours planned and expended, while a staffing measure accounts for quantity and types of personnel needed to do the work. This metric assists the Government in determining whether the developer has scheduled a sufficient number of employees to produce the product in the time and budget allotted.

b. Application.

(1) *Data collection and reporting.* Track for entire length of development and PDSS. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* The manpower metric reports the labor staffing in a software project. These elements include the planned level of effort, the actual level of effort, and the losses in the software staff measured by labor category. Anticipated effort and staffing profiles are derived from planning documents, usually the developer's proposal, software development plan or CRLCMP.

c. Data definitions. For each CSCI, labor category, and experience level tracked, collect—

(1) Labor category.

(2) Experience level (experienced, special, total).

(3) Number of personnel planned to be on staff for the reporting period.

(4) Number of personnel actually on staff in the reporting period.

(5) Number of unplanned losses in personnel that occurred.

(6) Number of labor hours planned to be expended in the reporting period (cumulative).

(7) Number of labor hours actually expended in the reporting period (cumulative).

d. Presentation and analysis. The primary information obtained from the manpower metric is derived by comparing planned and actual levels of effort and personnel. Figure 10-24 depicts the effort measure for an entire system for all labor categories over time. Figure 10-25 is an example of a staffing profile. Displays can be organized by CSCI or individual labor category for more detailed analysis.

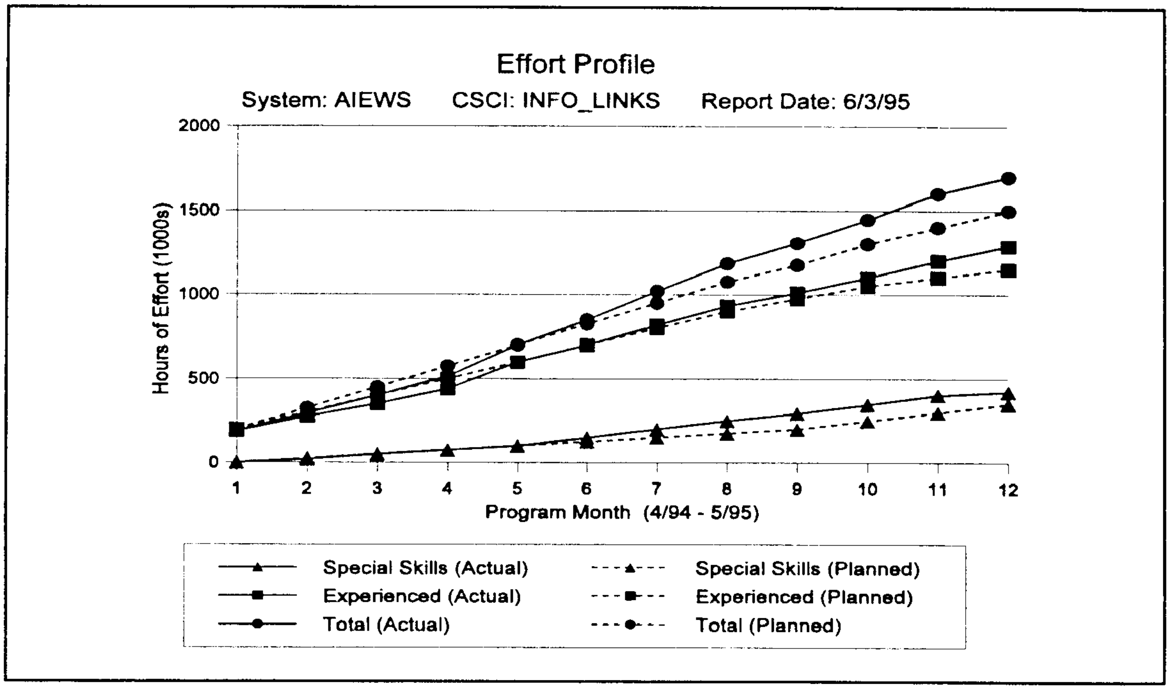


Figure 10-24. Sample graph of manpower effort measure

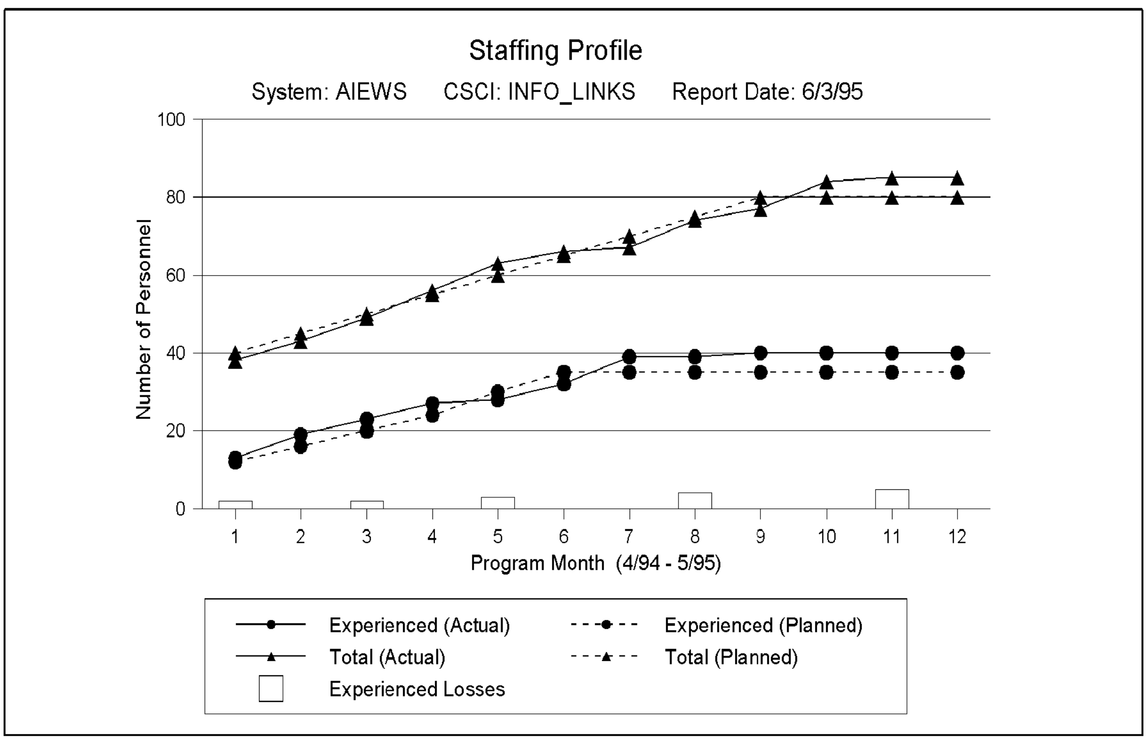


Figure 10-25. Sample graph of manpower staffing profile

e. Management information.

(1) Software staff includes those engineering and management personnel directly involved with any software activity. Losses and gains for each labor category should be tracked to indicate potential problem areas. High turnover of key and experienced personnel can adversely affect project success. Adding many unplanned personnel late in the development process may indicate impending problems.

(2) Significant deviations from planned staffing levels may indicate problems in the developer's management procedures or problems in product quality that require additional effort to repair.

(3) The shape of the staffing profile curve tends to start at a moderate level at the beginning of a project, grow through design, peak at implementation and testing and diminish near the completion of integration testing. Individual labor categories, however, are likely to peak at different points in the life cycle. Any significant deviation between actual and planned values should be investigated to determine the cause. During PDSS, the staffing curve is typically flatter.

(4) The manpower metric is used primarily for project management and does not necessarily have a direct relationship with other technical and maturity metrics. For example, growth in number of personnel is not necessarily reflected by an increase in product quality.

(5) The relation of manpower with other metrics is shown in table 10-25.

Table 10–25
Manpower relation with other metrics

Metric	Relation
Cost, Schedule	Manpower is a primary driver of cost and schedule.
SEE	Developers with higher SEE ratings should be less sensitive to personnel turnover.
Requirements stability	Effort in analysis and test design should increase in response to significant requirements changes to provide additional support for modifying requirements and associated tests.
Design stability	Design and implementation effort should grow in response to significant design changes to provide additional support for design and code changes.
Fault profiles	Effort in areas affected by faults should increase in response to significant increases in the number of new faults to provide additional support to fix the problems.
Development progress	Significant changes in staffing or effort allocation will likely be reflected by changes in planned and/or actual levels of development progress.

f. Tailoring. Depending on the planned level of maintenance, it may be expedient to collect and report the manpower metric only at the system level during PDSS, rather than by CSCI.

10–20. Development progress metric

a. Description. The development progress metric measures the degree of completeness of the software development effort, indicating the readiness to proceed to subsequent activities in software development.

b. Application.

(1) *Data collection and reporting.* Begin collecting during software requirements analysis and continue throughout software development and PDSS. The recommended reporting frequency for this metric is monthly.

(2) *Preparation.* Schedules for software unit development, test, and integration are needed.

c. Data definitions.

(1) For each CSCI collect—

(a) Number of software units in the CSCI.

(b) Number of units planned and actual number of units fully designed and reviewed by the Government (cumulative).

(c) Number of units planned and actual number of units fully coded and successfully unit tested (cumulative).

(d) Number of units planned and actual number of units fully integrated into the CSCI (cumulative).

(2) " Successfully" tested is defined as completing all test cases (required test coverage) with no defects." Integrated" is defined as being actually and logically connected (in a static sense) with all required units. Dynamic tasking is not considered when determining if units are integrated.

d. Presentation and analysis. The basic information obtained by the development progress metric is derived by comparing planned to actual quantities of software units that have completed various steps in the development process over time. Typically, degrees of design, implementation, and test are tracked. A sample display of development progress is shown in figure 10–26. To enhance readability, it is recommended to plot the planned and actual numbers of units as percentages of the total number of units.

e. Management information.

(1) The three development steps at which counts of units are taken in this metric are for those units that have completed the software design, software implementation and unit test, and unit integration and testing activities.

(2) Design, coding and unit testing, and integration of units should progress at a reasonable rate. Examining the progress in these three categories versus what was originally planned can indicate potential problems with schedule and cost.

(3) In certain instances, consideration should be given to a possible re-baseline of the software, such as in an evolutionary approach. That is, development progress may appear to suddenly degrade because the overall set of requirements (and projected number of total units) has been expanded to cover another evolution of user requirements. You can simply add to the total number of units due to changes in the requirements or begin tracking the newer evolution or build separately. In either case, all other metrics would need to reflect the re-baseline as well.

(4) You cannot judge whether the objectives of the development plan can be achieved using only the development progress metric. The testing metrics are needed as well.

(5) The relation of development progress with other metrics is shown in table 10–26.

Table 10–26
Development progress relation with other metrics

Metric	Relation
Cost, Schedule	Cost and Schedule will be adversely affected by significant delays shown by development progress.
CRU	Are any target upper bounds being approached or capacities exceeded as more units are integrated?
Requirements traceability	Are units that implement critical or high priority user functions progressing as expected?
Requirements stability	Development progress measures should decrease in response to significant requirements changes as units undergo redesign and retest or new units are added to the total.
Complexity	Are highly complex units progressing through the development activities at a reasonable rate?
Depth of testing	Are depth measure coverages reasonable for units that have completed unit testing?
Fault profiles	Progress may lag behind plan if unusually large numbers of problems are being corrected. Conversely, reasonable progress with many outstanding faults may indicate corrective action is not occurring.
Manpower	Are delays in progress due to insufficient staffing or high turnover of personnel?

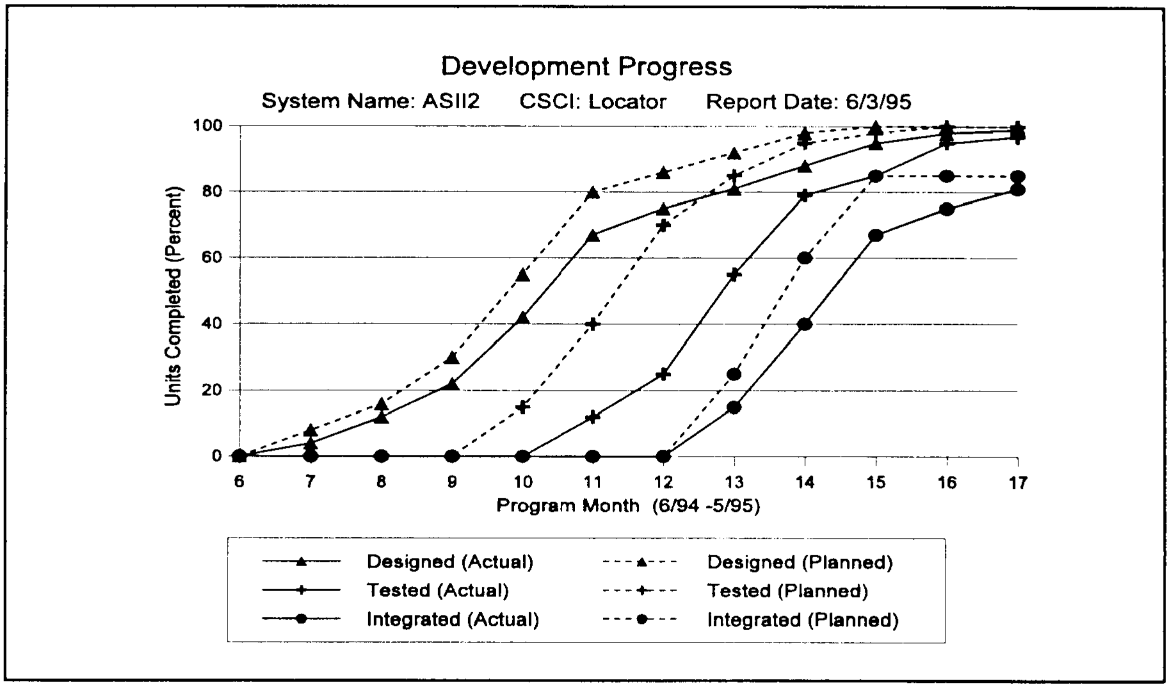


Figure 10-26. Sample graph of development progress

f. Tailoring.

(1) Use local data and local data formats as input to the calculations or as input to tools used for the calculations.

(2) Different data definitions may be more appropriate for software engineering environments using object-oriented or fourth generation languages. Tracking the progress of objects, function points, standard data elements, or other entities through comparable development steps may be more meaningful.

(3) It may be worthwhile to define and track CSCI-level attributes with development progress, such as number of CSCIs that have completed software qualification testing and number that have completed CSCI/HWCI integration, particularly for large systems or those with many interfaces to other systems.

Section III
Relating Metrics to Management Issues

10–21. MAIS assessment illustration

Paragraph 8–9 discussed the requirements of DOD 5000.2–R and quarterly status reporting. While the metrics in this chapter do not fully address all the reporting issues, they can provide useful information as part of the total program input for each assessment area. Table 10–27 relates the Army metrics to each DOD issue.

Table 10–27
Metric correlation to MAIS status report requirements

MAIS assessment issue	Assessment must address at least	Contributing metric
Schedule and progress	Completion of program milestones, significant events, and individual work items	Cost, Schedule, Development progress
Growth and stability	Stability of required functionality or capability, and the volume of software delivered to provide required capability	Requirements traceability, Requirements stability, Design stability, Complexity, Computer resource utilization
Funding and personnel resources	The balance between work to be performed and resources assigned and used	Cost, Manpower
Product quality	The ability of delivered product to support the user's need without failure, and problems and errors discovered during testing that result in the need for rework	Fault Profiles, Reliability, Complexity, Breadth of testing, Depth of testing
Software development performance	The developer's productivity capabilities relative to program needs	Software Engineering Environment, Development progress
Technical adequacy	Software reuse, use of Ada for software development, and use of approved standard data elements	Requirements traceability, Development progress, Complexity

Appendix A
References

Section I
Required Publications

AR 25–3
Army Life Cycle Management of Information Systems. (Cited in paras 1–4, 3–5, 5–6, 5–8, 5–9.)

AR 70–1
Army Acquisition Policy. (Cited in paras 1–4, 4–6, 5–6, 5–9)

AR 73–1
Test and Evaluation Policy. (Cited in paras 1–1, 1–4, 1–6, 1–9, 1–10, 2–2, 3–2, 4–3, 5–8, 5–10, 6–39, 6–41, 6–46, 6–47, 6–54.)

AR 380–19
Information Systems Security. (Cited in para 3–2, 3–8.)

AR 700–142
Instructions for Materiel Release, Fielding, and Transfer. (Cited in para 7–6, 7–11.)

DA Pam 70–3
Army Acquisition Procedures. (Cited in paras 4–6, 5–9.)

MIL–STD–498
Software Development and Documentation. (Cited in paras 1–13, 2–2, 10–11.)

Section II
Related Publications

A related publication is merely a source of additional information. The user does not have to read it to understand this publication.

AR 5–11
Modeling and Simulation

AR 40–60
Policies and Procedures for the Acquisition of Medical Materiel

AR 70–1
Systems Acquisition Policy and Procedures

AR 71–2
Basis of Issue Plans (BOIP) and Qualitative and Quantitative Personnel Requirements Information (QQPRI)

AR 71–9
Materiel Objectives and Requirements

AR 385–16
System Safety Engineering and Management

AR 525–1
Strategic Systems

AR 602–2
Manpower and Personnel Integration (MANPRINT) in the Materiel Acquisition Process

AR 700–127
Integrated Logistic Support

CMU/SEI–87–TR–23
A Method for Assessing the Software Engineering Capability of Contractors

CMU/SEI–93–TR–24
Capability Maturity Model for Software (Version 1.1)

DA Pam 25–6
Configuration Management for Automated Information Systems

DA Pam 73–1
Test and Evaluation in Support of System Acquisition

DA Pam 73–2
Test and Evaluation Master Plan (TEMP) Format, Review and Approval Procedures

DA Pam 73–3
Critical Operational Issues and Criteria (COIC) Procedures and Guidelines

DA Pam 73–4
Developmental Test and Evaluation (DT&E) Guidelines

DA Pam 73–5
Operational Test and Evaluation (OT&E) Guidelines

DA Pam 73–8
Critical Elements in Support of Test and Evaluation

DA Pam 700–55
Instructions for Preparing the ILSP

DA Pam 700–142
Instructions for Materiel Release, Fielding, and Transfer

DOD 5000.2–R
Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs

DODD 3405.1
Computer Programming Language Policy

DODD 5000.1
Defense Acquisition

DODD 8000.1
Defense Information Management Program

DODI 7000.2
Cost/Schedule Control System Criteria

DOD–STD–2167A
Defense System Software Development

DOD–STD–7935A
DOD Automated Information System (AIS) Documentation Standards

MIL–HDBK–881
Work Breakdown Structure for Defense Materiel Items

MIL–HDBK–245C
Preparation of Statement of Work (SOW)

MIL–STD–973
Configuration Management

NBS 500–99
Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric

NSWCDD 84–373
Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS)

TRADOC Regulation 71–2
Combat Developments for Battlefield Automated Systems

TRADOC Pamphlet 71–7

Operational Requirements for Battlefield Automated Systems

Section III
Prescribed Forms
This section contains no entries.

Section IV
Referenced Forms

DA Form 5005–R
Engineering Change Proposal - Software

DD Form 1423
Contract Data Requirements List

Appendix B
Statement of Work (SOW) Considerations

B–1. General

a. The statement of work (SOW) defines all work tasks and services to be performed over the course of a Government contract. It provides information which cannot be defined in the limited scope of technical specifications and contract data requirements lists (CDRLs). Specifications are limited to descriptions of technical and performance requirements of products. CDRL items are limited to describing technical data to be delivered.

b. Software test and evaluation is an emerging technology for which specific requirements are not well defined in existing specifications and standards. Specific tasks for planning and executing software T&E should be tailored to the technical and management characteristics of each software development. MIL-HDBK–245C establishes formal requirements for developing and implementing a SOW.

c. The SOW is part of a binding legal document, a contract, and should be prepared carefully and accurately.

d. The material in this appendix can be tailored and applied to Government in-house work agreements involving software development and testing. Substitute the name of the Government agency or the term “ developer” for “ contractor” in the sample paragraphs.

B–2. T&E involvement in the solicitation process

a. In general, the Government acquires the products and services described in this pamphlet by means of formal solicitations. The solicitation process is typically comprised of three major steps:

(1) The Government makes known the products or services needed, with applicable conditions, by issuing a request for proposal (RFP).

(2) Industry responses to provide the items in the RFP are evaluated and the most qualified organization(s) identified to perform the work.

(3) Final negotiation of terms between the Government and industry occurs and one or more contracts are awarded.

b. In order to have an impact on specific software development tasks, software T&E personnel should be involved in developing portions of the RFP, including its SOW. Specific items which should be addressed by software T&E personnel are:

(1) Distribution of CDRL items on a DD Form 1423 (Contract Data Requirements List) should include independent evaluators, the PM’s software matrix support activity, and LCSEC/PDSS personnel.

(2) Software tests should permit derivation of data which support stated software maturity measurements and the selected metrics set.

(3) Contracted software testing should provide usable data for Government evaluations.

c. Some specific software T&E issues which should be addressed in contractor proposals to SOW items for procuring software are provided in figure B–1. Checklist items with a subjective score of four or less should be clarified or elaborated.

B–3. Acquiring metrics information

Sample paragraphs regarding metric data for use in CE are provided for reference in figure B–2. The solicitation sections to which the paragraphs apply are noted. Select and tailor only those paragraphs that are most applicable.

Software T&E Issue	Rating
1. Flow down of software T&E requirements to third party development activities and subcontractors.	10
2. Data rights and access to software T&E data.	10
3. Relationship of software T&E to the software quality process.	10
4. Requirements for demonstration and evaluation of software maintainability.	10
5. Interface of software T&E to configuration management/control activities.	10
6. Documentation and utilization of software problem/change reports identified during software T&E.	5
7. Delivery of and utilization of software development documentation by software T&E activities.	5
8. Involvement of software T&E in the review and audit process.	10
9. Software T&E issues in selection of the developer (SEE metric rating).	10
10. Requirements for software test hooks or data ports for software T&E instrumentation.	10
11. Coordination of IV&V activities with software T&E.	10
12. Reporting of correlation between software test methods and objectives of the software T&E program.	10
13. Requirements for system and software specifications to identify testable requirements.	10
14. Scheduling of levels of test to support evaluation of software maturity.	10
15. Independent audits are conducted for each step of the software development process.	10
16. Standards are established for T&E of existing designs and code for re-use in new applications.	10
17. Standards are applied to the preparation of unit test cases.	10
18. Coding standards are used.	[illegible]
19. Statistics on software design errors are available to support T&E.	10
20. Metrics are used and applied logically and usefully.	[illegible]
21. Statistics on software implementation and test errors are plotted over time to show trends.	10
22. Capacity requirements (CRU) are monitored for computer memory utilization.	10
23. Capacity requirements are monitored for CPU throughput and utilization.	10
24. Capacity requirements are monitored for I/O channel utilization.	10
25. Security levels and security accreditation T&E is addressed.	10
26. Test coverage is measured and recorded for each level of software testing.	10
27. Action items resulting from reviews are coordinated with T&E program and data results.	10
28. Software problem/change reports are prioritized, recorded and delivered to the Government.	10
29. A formal mechanism is used for controlling changes to code.	5
30. Identification of models/simulations used for T&E purposes.	10
31. A formal mechanism is used for configuration management of the software tools used in T&E.	[illegible]
32. Testbeds and/or instrumentation needs are identified or required as part of the T&E program.	10
33. Validation or certification of testbeds or instrumentation is required prior to T&E use.	10
34. Interoperability standards are invoked and tests of corresponding interfaces adequately addressed.	10
Indicate the level at which each issue has been addressed in the technical proposal. Assign a score between 0 (not addressed) and 10 (very well addressed).	

Figure B-1. Software T&E issue checklist example

Section C - Description/specifications/work statement

3.n.1 Software development risk control procedures. The contractor shall document his software risk control procedures to ensure the effort is being carried out according to the technical and management objectives for the program. The measures or metrics used to monitor and identify risk in schedule and progress, growth and stability, funding and personnel resources, product quality, software development performance, and technical adequacy shall be described, as well as how each measure will be used to support its objective. The technical or management issues to be addressed, the measures to be utilized, definitions of specified measures and measurement methodologies, and description of how measurement information will be used in managing the contractor's program shall be included. The contractor shall provide a schedule identifying an appropriate starting date and collection frequency for each measure that maximizes its contribution to risk management.

3.n.1.1 Deliverable product. The contractor shall prepare the software risk control procedures in contractor format.

3.n.1.2 Schedule. The contractor shall submit the software risk control procedures IAW the schedule specified in DD Form 1423.

3.n.2 Software risk control records. The contractor shall maintain and use any record or data essential to the economic and effective operation of his software development and maintenance program. These records shall be available for review by Government personnel and copies of individual records shall be furnished to the Government as specified in paragraph *3.n.2.1.*

3.n.2.1 Deliverable product. The contractor shall provide the data comprising the software measures identified in paragraph *3.n.1* in contractor format.

3.n.2.2 Schedule. The contractor shall submit software risk control records IAW the schedule specified in DD Form 1423.

Section L - Instructions, conditions, and notices to offerors

Understanding of software development process and software risk measures. The offeror shall demonstrate a thorough understanding of the technical and program management approaches for building and maintaining quality software for the XYZ program. In doing so, the identification and constructive use of appropriate objective measurements in assessing and ensuring software product and process quality should be addressed. It should be noted that for each work area the offeror shall propose what guidance document (e.g. military or non-military standard or specification and/or contractor generated guidance) will be followed in performing the effort.

Past performance. The offeror shall demonstrate that experience is resident to permit accomplishing the required software development and maintenance effort without a learning and education period (other than particulars on the specific programs or problems involved). This shall be done by describing past work of similar or identical nature in such a manner that an evaluation can be made on the relevance of this experience to the requirements of this solicitation. The offeror shall select three to five contracts which may be from contracts with Federal, State or local Government or from contracts with private firms, by which to demonstrate it's past performance. These contracts shall, as nearly as possible, satisfy the following criteria:

a. The past performance information should be relevant and comparable, in scope, domain, and complexity, to the work being performed under the contract.
b. Demonstrate the application and use of software measures or metrics as successful risk reduction factors in performing the past effort. Applicable guidance documents used in developing or administering a software metrics program shall be identified. Guidance that has proved useful in the past can be found in DA Pamphlet 73-7, *Army Software Test and Evaluation Guidelines.*

Section M - Evaluation factors for award

The past performance of all technically acceptable proposals will be considered.

Figure B-2. Sample metrics paragraphs

Appendix C
Metrics Data Collection Templates

C–1. General
This appendix contains sample templates for uniform collection of the metrics described in chapter 10 of this pamphlet. The needs and resources of each program will determine the precise data submission start times, reporting frequency and data elements for each metric. The templates can be tailored or used as-is, in conjunction with appendix B.

C–2. Organization
A template is provided for each metric described in chapter 10. Templates consist of one or more records. Each data element in a metric record is identified with a capitalized mnemonic to simplify cross referencing between its text definition and format description.

C–3. Cost metric template
One type of data record is used to describe cost metric data. For each activity type, a cost metric data record contains the following information. The format for cost metric data is shown in table C–1. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies. No value is required for this field when reporting system/project level costs.

e. ACTIVITY_TYPE - The type of effort or product associated with the collected data. Examples of activity types for CSCI level reporting and system level reporting are given in chapter 10 of DA Pam 73–7.

f. BCWS - Budgeted cost of work scheduled (cumulative to date). The total number of dollars that had been budgeted for the work scheduled to be accomplished for the CSCI as of this reporting period.

g. BCWP - Budgeted cost of work performed (cumulative to date). The total number of dollars budgeted for the work actually performed on the CSCI as of this reporting period.

h. ACWP - Actual cost of work performed (cumulative to date). The total number of dollars which was actually spent for the work done on the CSCI as of this reporting period.

C–4. Schedule metric template
One type of data record is used to describe schedule metric data. For each milestone, deliverable, event or activity, a schedule metric data record contains the following information. The format for schedule metric data is shown in table C–2. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies, if any. This field should not contain a value when reporting non-CSCI specific events.

e. EVENT_TYPE - The kind of event that categorizes EVENT_NAME. Examples are SW DEV ACTIVITY, DOCT DELIV, FORMAL REVIEW, representing software development activity, documentation delivery, and formal review.

f. EVENT_NAME - The name of the milestone, deliverable, event or activity this record describes. Examples of events are formal system reviews, testing events, and software data product deliveries. Specific examples are SW REQTS ANAL, CSCI QUAL TESTG, CSCI/HWCI INTEG, PCA, SRS DRAFT representing software development activities: software requirements analysis, CSCI qualification testing and CSCI/HWCI integration and testing; physical configuration audit and software requirements specification delivery.

g. PLAN_START_DATE - The date on which the event is planned to start.

h. PLAN_END_DATE - The date on which the event is planned to be completed.

i. ACTUAL_START_DATE - The date the event actually began.

j. ACTUAL_END_DATE - The date the event actually completed.

Table C–1
Cost metric data record format

Data element Name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
ACTIVITY_TYPE	C	15				
BCWS	N	12	2	Dollars	0.00	999999999.99
BCWP	N	12	2	Dollars	0.00	999999999.99
ACWP	N	12	2	Dollars	0.00	999999999.99

Note: The following standard definitions apply to all data record format tables in this appendix.

Data Type	C	(Character) Consists of alphabetic characters, numeric characters, or symbols. An embedded blank between two non-blank characters is an acceptable character. *Note:* Numeric characters, also called numerals, cannot be used directly in mathematical calculations.
	N	(Numeric) Numerals (0,1,2, ... 9), decimal point or negative sign.
Decimal		Number of numerals after the decimal point.
Units	Date	A legitimate date expressed as MM/DD/YYYY designating month/day/year.

Table C–2
Schedule metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
EVENT_TYPE	C	15				
EVENT_NAME	C	20				
PLAN_START_DATE	C	10		Date		
PLAN_END_DATE	C	10		Date		
ACTUAL_START_DATE	C	10		Date		
ACTUAL_END_DATE	C	10		Date		

C–5. Computer resource utilization (CRU) metric template

a. One type of data record is used to describe computer resource utilization metric data. The record may report either hardware resource (device) utilization data or software resource allocation data.

b. In hardware resource utilization reporting, for each computer resource in the system, a CRU metric record contains the following information. The format for utilization data is shown in table C–3. The record consists of data elements:

(1) DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

(2) SYSTEM_NAME - Name of the system to which this data applies.

(3) BUILD_ID - Build, block or version identifier to which this data applies.

(4) CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies, if any. This field should not contain a value when reporting hardware resource utilization data.

(5) RESOURCE_ID - A unique identifier for the resource. No value is required for this field if reporting total CSCI software allocation data for a single RESOURCE_TYPE.

(6) RESOURCE_TYPE - The kind of resource represented by RESOURCE_ID. Examples are CPU, RAM, I/O channel, disk storage and LAN.

(7) UNITS_OF_MEASURE - The type of measurement units that the resource's capacity is expressed in. For example, megabytes or millions of instructions per second.

(8) RESOURCE_CAPACITY - The number of UNITS_OF_MEASURE that represents the total capacity (100%) of the resource.

(9) PCNT_TGT_UPR_BND - The target upper bound utilization value for the resource. This is the desired maximum value for this resource's utilization expressed as a percentage of total capacity of the resource.

(10) PCNT_PROJECTED - The projected capacity utilization value for the resource. This is the estimated percentage of maximum utilization expected at delivery.

(11) PCNT_ACTUAL - The measured value of resource capacity utilized during peak operational loading periods expressed as a percentage of total capacity of the resource.

(12) COMP_TYPE - Identification as to whether measurements were taken on the actual target machine or a software test environment configuration.

c. In software resource allocation reporting, for each CSCI in the system, a CRU metric record contains the information in data elements (1) through (12) above. The format for utilization data is shown in table C–3. If reporting CSCI allocations to individual hardware devices, supply values for both RESOURCE_ID and RESOURCE_TYPE.

Table C–3
CRU metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
RESOURCE_ID	C	12				
RESOURCE_TYPE	C	12				
UNITS_OF_MEASURE	C	12				
RESOURCE_CAPACITY	N	11	2	**	0	99999999.99
PCNT_TGT_UPR_BND	N	3	0	Percent	0	100
PCNT_PROJECTED	N	3	0	Percent	0	100
PCNT_ACTUAL	N	3	0	Percent	0	100
COMP_TYPE	C	1			*	*

Notes:

* Range of values is limited to H, T.

** Express RESOURCE values in UNITS_OF_MEASURE.

C–6. Software Engineering Environment (SEE) metric template

One type of data record is used to described SEE metric data. For each developer on which a software maturity level assessment is performed, a SEE metric record contains the following information. The format for software engineering environment metric data is shown in table C–4. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. DEVELOPER_NAME - The name of the developer that was evaluated.

c. SYSTEM_NAME - Name of the system to which this data applies.

d. BUILD_ID - Build, block or version identifier to which this data applies.

e. MATURITY_LEVEL - The overall process maturity level assigned to the developer.

f. MATURITY_DATE - The date the process maturity level was assigned to the developer.

g. KEY_PROCESS_AREA - The SEI key process area (KPA) whose results are reported by this record. See chapter 10 of DA Pam 73–7 for a list of applicable KPAs.

h. KPA_RESULT - Key process area KEY_PROCESS_AREA was assessed as satisfactory, unsatisfactory or not rated. Not rated means the KPA was not reviewed in this assessment.

i. ASSESSMENT_TYPE - Indicator as to whether the assessment was performed by the developer as a self-assessment, the acquirer or an authorized representative representing the sponsor, or by an independent third party organization.

Table C–4
SEE metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
DEVELOPER_NAME	C	15				
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
MATURITY_LEVEL	N	1	0		1	5
MATURITY_DATE	C	10		Date		
KEY_PROCESS_AREA	C	15			*	*
KPA_RESULT	C	1			**	**
ASSESSMENT_TYPE	C	6			***	***

Notes:

* Range of values is limited to SW CM, SW QA, SW SUB MGT, SW PROJ TRACK, SW PROJ PLAN, REQTS MGT, PEER REVIEW, GROUP COOR, SW PROD ENGR, INTEG SW MGT, TRAINING PGM, PROCESS DEF, PROCESS FOCUS, QUAL MGT, PROCESS ANAL, PROC CHNG MGT, TECH INNOV, DEFECT PREV.

** Range of values is limited to S, U, N.

*** Range of values is limited to SELF, ACQ, INDEP.

C–7. Requirements traceability metric template

a. One type of data record is used to describe requirements traceability metric data. The record may report either software requirements traceability results or overall requirements traceability results.

b. Software requirements traceability reporting. For each CSCI in the system, a requirements traceability record contains the following information. The format for requirements traceability metric data is shown in table C–5. The record consists of data elements:

(1) DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

(2) SYSTEM_NAME - Name of the system to which this data applies.

(3) BUILD_ID - Build, block or version identifier to which this data applies.

(4) CSCI_NAME - Name of the Computer Software Configuration Item (CSCI) to which the data applies, if any. If overall system or non-CSCI specific documents or requirements are being traced, there should be no value in this field.

(5) FROM_DOCT - The name of the document whose requirements were examined in order to them trace forward into the TO_DOCT.

(6) FROM_DOCT_VER_ID - The configuration identifier assigned to the edition of the FROM_DOCT that was examined.

(7) FROM_DOCT_TYPE - The type of document or type of requirements that were examined in the FROM_DOCT. Examples are SW REQTS, IF REQTS, USER REQTS, HW SW DESIGN, SW UNIT DESIGN, CODE, SW TEST CASES, and SYSTEM REQTS representing software requirements, interface requirements, user requirements, CSCI- wide and CSCI architectural design, software unit design, code, software test cases and system requirements respectively.

(8) FROM_DOCT_NUM_REQ - The number of FROM_DOCT_TYPE requirements allocated to document FROM_DOCT.

(9) TO_DOCT - The name of the document examined in order to find the appropriate link to the requirements in FROM_DOCT.

(10) TO_DOCT_VER_ID - The configuration identifier assigned to the edition of the TO_DOCT that was examined.

(11) TO_DOCT_TYPE - The type of document or type of requirements that were examined in the TO_DOCT. See examples in item (7).

(12) TO_DOCT_NUM_REQ - The number of TO_DOCT_TYPE requirements allocated to document TO_DOCT.

(13) TRACE_FROM_TO - The number of requirements in FROM_DOCT that were successfully traced into the TO_DOCT.

(14) NO_TRACE_FROM_TO - The number of requirements in FROM_DOCT that could not be traced into the TO_DOCT, but should have been addressed in TO_DOCT.

(15) BACK_TRACE_TO_FROM - The number of requirements in TO_DOCT that were successfully traced backward to the FROM_DOCT.

(16) NO_BK_TRACE_TO_FROM - The number of requirements in TO_DOCT that did not trace back to the FROM_DOCT.

c. Overall requirements traceability reporting. For each set of non-CSCI specific documents examined, a requirements traceability metric record contains the information in data elements (1) through (3) and (5) through (16). The format for traceability data is shown in table C–5.

Table C–5
Requirements traceability data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
FROM_DOCT	C	30				
FROM_DOCT_VER_ID	C	15				
FROM_DOCT_TYPE	C	15				
FROM_DOCT_NUM_REQ	N	5	0		0	99999
TO_DOCT	C	30				
TO_DOCT_VER_ID	C	15				
TO_DOCT_TYPE	C	15				
TO_DOCT_NUM_REQ	N	5	0		0	99999
TRACE_FROM_TO	N	5	0		0	99999
NO_TRACE_FROM_TO	N	5	0		0	99999
BACK_TRACE_TO_FROM	N	5	0		0	99999
NO_BACK_TRACE_TO_FROM	N	5	0		0	99999

C–8. Requirements stability metric template

One type of data record is used to describe requirements stability metric data. For each CSCI in the system, a requirements stability metric record contains the following information. The format for requirements stability metric data is shown in table C–6. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. TOT_REQ_DISCREP - The total number of software requirements discrepancies detected to date (cumulative).

f. TOT_CLOSED_DISCREP - The total number of software requirements discrepancies to date (cumulative) which are closed as of this reporting period.

g. USER_ECP - The number of Engineering Change Proposals-Software (ECP-Ss) submitted in this reporting period by the user against software requirements.

h. USER_SLOC - The number of source lines of code affected in this reporting period by approved software requirements related ECP-Ss that were submitted by the user. Source lines of code are non-blank, non-comment, executable and data statements.

i. USER_MODS_AFFECT - The number of software units in the CSCI which are affected in this reporting period by approved software requirements-related ECP-Ss submitted by the user.

j. DEV_ECP - The number of ECP-Ss submitted in this reporting period by the developer against software requirements.

k. DEV_SLOC - The number of source lines of code affected in this reporting period by approved software requirements related ECP-Ss that were submitted by the developer. Source lines of code are non-blank, non-comment, executable and data statements.

l. DEV_MODS_AFFECT - The number of software units in the CSCI which are affected in this reporting period by approved software requirements-related ECP-Ss submitted by the developer.

m. SLOC - The number of source lines of code in the CSCI. Source lines of code are non-blank, non-comment, executable and data statements.

n. NUM_SRS_REQ - The number of Software Requirements Specification (SRS) requirements for this CSCI.

o. NUM_SRS_REQ_ADD - The number of SRS requirements added in this reporting period due to approved ECP-Ss.

p. NUM_SRS_REQ_MOD - The number of SRS requirements modified in this reporting period due to approved ECP-Ss.

q. NUM_SRS_REQ_DEL - The number of SRS requirements deleted in this reporting period due to approved ECP-Ss.

Table C–6
Requirements stability metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
TOT_REQ_DISCREP	N	5	0		0	99999
TOT_CLOSED_DISCREP	N	5	0		0	99999
USER_ECP	N	5	0		0	99999
USER_SLOC	N	7	0		0	9999999
USER_MODS_AFFECT	N	5	0		0	99999
DEV_ECP	N	5	0		0	99999
DEV_SLOC	N	7	0		0	9999999
DEV_MODS_AFFECT	N	5	0		0	99999
SLOC	N	7	0		0	9999999
NUM_SRS_REQ	N	5	0		0	99999
NUM_SRS_REQ_ADD	N	5	0		0	99999
NUM_SRS_REQ_MOD	N	5	0		0	99999
NUM_SRS_REQ_DEL	N	5	0		0	99999

C–9. Design stability metric template

One type of data record is used to describe design stability metric data. For each CSCI in each version delivered during this reporting period, a design stability metric record contains the following information. The format for design stability metric data is shown in table C–7. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. VERSION_ID - The configuration control version identification of the CSCI that is being reported.

f. COMP_DATE - The planned date that VERSION_ID will be completed.

g. TOTMOD_FINAL - The total number of software units that are planned to comprise the final delivery of the CSCI.

h. TOTMOD_N_DESIGN - The number of units comprising the CSCI delivered in VERSION_ID.

i. NUM_MOD_CHANGED - The number of units in which design related changes were made since the last delivery of the CSCI.

j. NUM_MOD_ADDED - The number of units that were added to the CSCI since the last delivery.

k. NUM_MOD_DELETED - The number of units that were deleted from the CSCI since the last delivery.

Table C–7
Design stability metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
VERSION_ID	C	15				
COMP_DATE	C	10		Date		
TOTMOD_FINAL	N	5	0		0	99999
TOTMOD_N_DESIGN	N	5	0		0	99999
NUM_MOD_CHANGED	N	5	0		0	99999
NUM_MOD_ADDED	N	5	0		0	99999
NUM_MOD_DELETED	N	5	0		0	99999

C–10. Complexity metric template

One type of data record is used to describe complexity metric data. For each software unit in the system that has been added, modified, or deleted, a complexity metric record contains the following information. The format for complexity metric data is shown in table C–8. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. UNIT_NAME - The name of the software unit to which the data applies.

f. DELETED - An indicator that this unit has been deleted from the CSCI. Acceptable values are Y (deleted) or N (unit is part of the system configuration).

g. LANGUAGE - The programming language the software unit is written in.

h. CYCLOMATIC_COMPLEX - The computed value of McCabe's cyclomatic complexity for the unit.

i. HALSTEAD_VOCAB - The computed value of Halstead's vocabulary term for the unit.

j. HALSTEAD_PGM_LENGTH - The computed value of Halstead's program length term for the unit.

k. HALSTEAD_VOLUME - The computed value of Halstead's program volume term for the unit.

l. CTRL_PATH_CROSS - The total number of occurrences in the unit where control paths cross.

m. SLOC - The total number of source lines of code in the unit. Source lines of code are non-blank, non-comment, executable and data statements.

n. PCNT_COMMENT - The computed percentage of comment lines in the unit.

o. PDL_OR_CODE - An indicator as to whether the complexity for this unit was computed on its Program Design Language (PDL) or source code representation.

Table C–8
Complexity metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
UNIT_NAME	C	15				
DELETED	C	1			*	*
LANGUAGE	C	12				
CYCLOMATIC_COMPLEX	N	5	0		0	99999
HALSTEAD_VOCAB	N	5	0		0	99999

Table C–8
Complexity metric data record format—Continued

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
HALSTEAD_PGM_LENGTH	N	5	0		0	99999
HALSTEAD_VOLUME	N	8	2		0.00	99999.99
CTRL_PATH_CROSS	N	5	0		0	99999
SLOC	N	5	0		0	99999
PCNT_COMMENT	N	3	0	Percent	0	100
PDL_OR_CODE	C	4			**	**

Notes:
* Range of values is limited to Y, N.
** Range of values is limited to PDL, CODE.

C–11. Breadth of testing metric template

One type of data record is used to describe breadth of testing metric data. For each CSCI in the system, a breadth of testing metric record contains the following information. The format for breadth of testing metric data is shown in table C–9. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. REQTS_TYPE - Indicator as to whether the requirements reported in this record are SRS requirements, IRS requirements or UFD requirements.

f. REQTS_PRIORITY - Level of priority (criticality) assigned to the requirements, if any.

g. NUM_REQTS - The total number of REQTS_TYPE requirements allocated to the CSCI under development.

h. TOT_TESTED_REQTS - The total number of REQTS_TYPE requirements for the CSCI that have been tested using approved test cases.

i. TOT_PASSED_REQTS - The total number of REQTS_TYPE requirements for the CSCI that have been successfully demonstrated through testing.

j. TEST_ID - The type of testing or a test event identifier with which this data is associated. Examples of TEST_ID are DT (Government Developmental Test), OT (Operational Test) or CSCI QUAL (CSCI qualification).

Table C–9
Breadth of testing metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
REQTS_TYPE	C	12			*	*
REQTS_PRIORITY	N	5	0		0	5
NUM_REQTS	N	5	0		0	99999
TOT_TESTED_REQTS	N	5	0		0	99999
TOT_PASSED_REQTS	N	5	0		0	99999
TEST_ID	C	12				

Notes:
* Range of values is limited to SRS, IRS, UFD.

C–12. Depth of testing metric template

One type of data record is used to describe depth of testing metric data. For each software unit in the system that has been added, modified, tested, or deleted since the last reporting period, a depth of testing metric record contains the following information. The format for depth of testing metric data is shown in table C–10. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. UNIT_NAME - The name of the software unit to which the data applies.

f. DELETED - An indicator that this unit has been deleted from the CSCI. Acceptable values are Y (deleted) or N (unit is part of the system configuration).

g. DEPTH_MEASURE - Designates to which depth attribute the remaining data elements of the record apply. Acceptable values are PATH, STATEMENT, INPUT, DECISION PNT corresponding to paths, statements, input instances and decision points, respectively.

h. TOT_IN_UNIT - The total number of attributes of type DEPTH_MEASURE in the unit being reported.

i. TOT_TESTED - The total number of attributes of type DEPTH_MEASURE in the unit that have been tested using approved test cases.

j. TOT_PASSED - The total number of attributes of type DEPTH_MEASURE in the unit that have been successfully tested in accordance with the measure's criteria stated in chapter 10 of DA Pam 73–7.

Table C–10
Depth of testing metric data record

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10	Date			
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
UNIT_NAME	C	15				
DELETED	C	1			*	*
DEPTH_MEASURE	C	12			**	**
TOT_IN_UNIT	N	5	0		0	99999
TOT_TESTED	N	5	0		0	99999
TOT_PASSED	N	5	0		0	99999

Notes:

* Range of values is limited to Y, N.

** Range of values is limited to PATH, STATEMENT, INPUT, DECISION PNT.

C–13. Fault profiles metric template

One type of data record is used to describe fault profiles metric data. For each CSCI in the system, a fault profiles metric record contains the following information. The format for fault profiles metric data is shown in table C–11. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. FAULT_PRIORITY - The priority level of faults described by the remaining data elements in the record.

f. FAULT_CATEGORY - The category of faults described by the remaining data elements in the record.

g. TOT_FLTS_DETECTED - The total number of faults with priority value FAULT_PRIORITY detected to date.

h. TOT_FLTS_CLOSED - The total number of faults with priority value FAULT_PRIORITY that have been closed/resolved to date.

i. AVGOPEN_AGE - The average number of days a currently open fault with priority value FAULT_PRIORITY has remained open.

j. AVGCLOSE_AGE - The average number of days it took to close a priority FAULT_PRIORITY fault.

k. AVG_AGE - The average age, in days, of a priority FAULT_PRIORITY fault (both open and closed).

Table C–11
Fault profiles metric record metric data format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
FAULT_PRIORITY	N	5	0		0	5
FAULT_CATEGORY	C	15				
TOT_FLTS_DETECTED	N	6	0		0	999999
TOT_FLTS_CLOSED	N	6	0		0	999999
AVGOPEN_AGE	N	5	0	Days	0	99999
AVGCLOSE_AGE	N	5	0	Days	0	99999
AVG_AGE	N	5	0	Days	0	99999

C–14. Reliability metric template

One type of data record is used to describe reliability metric data. For each system test event for which system/software reliability data is measured, a reliability metric record contains the following information. The format for reliability metric data is shown in table C–12. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. TEST_ID - The type of testing or a test event identifier with which this data is associated. Examples of TEST_ID are RGT (Reliability Growth Test), DT (Government Developmental Test), or OT (Operational Test).

e. MEASURED_FAIL_RATE - The computed failure rate of the software as measured in testing expressed in failures per month.

f. PROJECTED_FAIL_RATE - The projected failure rate of the software for the reporting period as calculated by the reliability analysis model.

g. RELY_MODEL - The name of the analytical model used to calculate the PROJECTED_FAILURE_RATE.

h. FAIL_RATE_OBJECTIVE - The desired goal of the acceptable number of software failures.

i. SYS_REQ_MTBF - The required/specified value of mean time between mission failures caused by system hardware or software against which the value measured during the test is compared for compliance.

j. PEST_SYS_MTBF - The computed point estimate of mean time between mission failures caused by system hardware or software as measured during the test event (field TEST_ID).

k. LCB_SYS_MTBF - The calculated 80% lower confidence bound value of mean time between mission failures caused by system hardware or software as measured during the test event.

l. PEST_SW_MTBF - The computed point estimate of mean time between mission failures caused by software as measured during the test event.

m. LCB_SW_MTBF - The calculated 80% lower confidence bound value of mean time between mission failures caused by software as measured during the test event.

n. REQ_MEAN_RESTOR - The required/specified value of mean time to restore the system to operational status.

o. REQ_MEDN_RESTOR - The required/specified value of median time to restore the system to operational status.

p. REQ_MAX95_RESTOR - The required/specified maximum 95th percentile value of time to restore the system to operational status.

q. MEAN_RESTOR_SYS - The computed mean time to restore the system to operational condition.

r. MEDN_RESTOR_SYS - The computed median time to restore the system to operational condition.

s. MAX95_RESTOR_SYS - The computed maximum 95th percentile value of time to restore the system to operational condition.

Table C–12
Reliability metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
TEST_ID	C	12				
MEASURED_FAIL_RATE	N	8	2	Fpm *	0.00	99999.99
PROJECTED_FAIL_RATE	N	8	2	Fpm	0.00	99999.99
RELY_MODEL	C	15				
FAIL_RATE_OBJECTIVE	N	8	2	Fpm	0.00	99999.99
SYS_REQ_MTBF	N	8	2	Hours	0.00	99999.99
PEST_SYS_MTBF	N	8	2	Hours	0.00	99999.99
LCB_SYS_MTBF	N	8	2	Hours	0.00	99999.99
PEST_SW_MTBF	N	8	2	Hours	0.00	99999.99
LCB_SW_MTBF	N	8	2	Hours	0.00	99999.99
REQ_MEAN_RESTOR	N	8	2	Hours	0.00	99999.99
REQ_MEDN_RESTOR	N	8	2	Hours	0.00	99999.99
REQ_MAX95_RESTOR	N	8	2	Hours	0.00	99999.99
MEAN_RESTOR_SYS	N	8	2	Hours	0.00	99999.99
MEDN_RESTOR_SYS	N	8	2	Hours	0.00	99999.99
MAX95_RESTOR_SYS	N	8	2	Hours	0.00	99999.99

Notes:
* Fpm - Failures per month.

C–15. Manpower metric template

One type of data record is used to describe manpower metric data. For each labor category and experience level reported, a manpower metric record contains the following information. The format for reliability metric data is shown in table C–13. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. LABOR_CATEGORY - The name of the labor category to which this data applies.

f. EXPERIENCE_LEVEL - The level of experience in the LABOR_CATEGORY to which this data applies.

g. NUM_STAFF_PLANNED - The number of personnel planned to be on staff in the reporting period.

h. NUM_STAFF_ACTUAL - The number of personnel actually on staff in the reporting period.

i. NUM_STAFF_LOSS - The number of unplanned losses in personnel that occurred in the reporting period.

j. TOT_HOURS_PLANNED - The total number of labor hours (cumulative to date) that are planned to be expended by the end of the reporting period.

k. TOT_HOURS_ACTUAL - The total number of labor hours (cumulative to date) that were actually expended in the reporting period.

Table C–13
Manpower metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
LABOR_CATEGORY	C	20				
EXPERIENCE_LEVEL	C	12				
NUM_STAFF_PLANNED	N	5	0		0	99999
NUM_STAFF_ACTUAL	N	5	0		0	99999

Table C–13
Manpower metric data record format—Continued

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
NUM_STAFF_LOSS	N	5	0		0	99999
TOT_HOURS_PLANNED	N	8	0	Hours	0	99999999
TOT_HOURS_ACTUAL	N	8	0	Hours	0	99999999

C–16. Development progress metric template

One type of data record is used to describe development progress metric data. For each CSCI in the system a development progress metric record contains the following information. The format for development progress metric data is shown in table C–14. The record consists of data elements:

a. DATA_DATE - The date associated with the values of the remaining data elements in the record. That is, the date when this data was current.

b. SYSTEM_NAME - Name of the system to which this data applies.

c. BUILD_ID - Build, block or version identifier to which this data applies.

d. CSCI_NAME - The Computer Software Configuration Item (CSCI) name to which the data applies.

e. TOT_UNITS_FINAL - The total number of software units planned for the final delivery of this CSCI in BUILD_ID.

f. TOT_DESIGNED_PLAN - The total number of units (cumulative to date) that are planned to have completed the design activity by the end of the reporting period.

g. TOT_UNIT_TESTED_PLAN - The total number of units (cumulative to date) that are planned to have completed implementation and unit testing by the end of the reporting period.

h. TOT_UNIT_INTEG_PLAN - The total number of units (cumulative to date) that are planned to have completed unit integration and testing by the end of the reporting period.

i. TOT_DESIGNED_ACTUAL - The total number of units (cumulative to date) that completed design through the current reporting period.

j. TOT_UNIT_TESTED_ACTUAL - The total number of units (cumulative to date) that completed implementation and unit testing through the current reporting period.

k. TOT_UNIT_INTEG_ACTUAL - The total number of units (cumulative to date) that completed unit integration and testing through the current reporting period.

Table C–14
Development progress metric data record format

Data element name	Data type	Width	Decimal	Units	Minimum value	Maximum value
DATA_DATE	C	10		Date		
SYSTEM_NAME	C	20				
BUILD_ID	C	15				
CSCI_NAME	C	15				
TOT_UNITS_FINAL	N	5	0		0	99999
TOT_DESIGNED_PLAN	N	5	0		0	99999
TOT_UNIT_TESTED_PLAN	N	5	0		0	99999
TOT_UNIT_INTEG_PLAN	N	5	0		0	99999
TOT_DESIGNED_ACTUAL	N	5	0		0	99999
TOT_UNIT_TESTED_ACTUAL	N	5	0		0	99999
TOT_UNIT_INTEG_ACTUAL	N	5	0		0	99999

Glossary

Section I
Abbreviations

ADP
automatic data processing

AMC
Army Materiel Command

AMCCOM
Armament, Munitions and Chemical Command

AMSAA
Army Materiel Systems Analysis Activity

AR
Army regulation

BAS
battlefield automated system

BOIP
basis of issue plan

CCB
configuration control board

CDA
Central Design Activity

CE
continuous evaluation

CECOM
Communications-Electronics Command

CI
configuration item

CM
configuration management

COM
computer operation manual

CPT
comparison test

CPU
central processing unit

CSC
computer software component

CSTA
Combat Systems Test Activity

DA
Department of the Army

DID
data item description

DOD
Department of Defense

DODD
Department of Defense directive

DODI
Department of Defense instruction

DS
database specification

DT
developmental test

DTP
detailed test plan

ECP
engineering change proposal

EDP
event design plan

EM
end user mannual

FDTE
force development testing and experimentation

I/O
input/output

IAW
in accordance with

IEP
independent evaluation plan

IER
independent evaluation report

ILSP
integrated logistic support plan

INSCOM
Intelligence and Security Command

IPT
integrated product team

IP
implementation procedures

IPR
in-process review

ISO
International Organization for Standardization

ISSC
Information Systems Support Command

JCS
Joint Chiefs of Staff

LOC
lines of code

MACOM
major Army command

MDR
milestone decision review

MM
maintenance manual

MOA
Memorandum of Agreement

MOE(s)
measure of effectiveness

MOP
measure of performance

MOT
multi-service operational test

MP
management plan

MS
milestone

MSC
major subordinate command

MTBF
mean-time-between-failure

MTTR
mean-time-to-repair

OEC
Operational Evaluation Command

OSUT
on-site user test

OT
operational test

PA
proponent agency

PCA
physical configuration audit

PCR
problem/change report

PEO
program executive officer

PM
program/project/product manager

PT
test plan

QA
quality assurance

QQPRI
qualitative and quantitative personnel requirements information

RAM
reliability, availability, maintainability random access memory

RFP
request for proposal

RT
test analysis report

SA
system assessment

SAT
software acceptance test

SCM
software configuration management

SCP
software change package

SDC
Software Development Center

SDD
software design description, software design document

SDF
software development file

SDP
software development plan

SEP
system evaluation plan

SIP
software installation plan

SIT
system integration test

SOW
statement of work

SPM
software programmer's manual

SPR
system post deployment review

SPS
software product specification

SQAP
software quality assurance plan

SQT
software qualification test

SS
system/subsystem specification

SSA
software support activity

SSEB
source selection evaluation board

SSS
system/subsystem specification

SUM
software user manual

TACOM
Tank Automotive Command

TDP
test design plan

TECOM
Test and Evaluation Command

TFT
technical feasibility testing

TIWG
Test Integration Working Group

TR
technical report

TRADOC
Training and Doctrine Command

UFD
users' functional description

WBS
work breakdown structure

Section II
Terms

Allocated baseline
The initially approved documentation describing an item's functional, interoperability, and interface characteristics that are allocated from those of a system or a higher level configuration item, interface requirements with interfacing configuration items, additional design constraints, and the verification required to demonstrate the achievement of those specified characteristics. (Reference MIL-STD–973)

Army technical architecture (ATA)
The approved reference that identifies mandated and recommended standards regarding information management processes and implementations for systems that perform computing and communications functions for the Army. The ATA incorporates elements of the DOD's technical architecture framework for information management (TAFIM) as well as other DOD acquisition and standardization initiatives.

Automated information system (AIS)
A combination of information, computer and telecommunications resources and other information technology and personnel resources that collects, records, processes, stores, communicates, retrieves, and displays information. (Reference AR 25–3)

Baseline
Configuration documentation formally designated and fixed at a specific time during a configuration item's life cycle. Configuration baselines, plus approved changes from those baselines constitute the current configuration. (Reference MIL-STD–973)

Benchmark test files (BMTF)
A database of known content against which a controlled set of inputs is processed and from which output results may be predicted. This term is used in reference to a test environment and pre-established test cases/data.

CASE tools
Computer aided software engineering (CASE) tools are systems for building systems; they automate elements of the requirements analysis, design, development or test process.

Code walk-through
The process of assessing the level of software performance and design structure that requires the developer to demonstrate the capabilities of the software to technical, functional, and user representatives.

Computer resources
The totality of computer personnel, documentation, services, and supplies applied to a given effort. This includes hardware, software, services, personnel, documentation and supplies.

Computer resource life cycle management plan (CRLCMP)
The primary Government planning document used at all decision levels for assessing the adequacy of the overall computer resources management efforts throughout a system's life. (Reference DA Pamphlet 70–3)

Computer resources IPT
Often established by the material developer after Milestone I for each AR 70–1 system to aid in the management of system computer resources. A computer resources IPT assists in insuring compliance with policy, procedures, plans and standards established for computer resources. (Reference AR 73–1)

Computer software configuration item (CSCI)
A configuration item that is software. (Reference MIL-STD–973)

Configuration item (CI)
An aggregation of hardware, software, or both that satisfies an end use function and is designated by the Government for separate configuration management. (Reference MIL-STD–973)

Configuration management
A discipline applying technical and administrative direction and surveillance to (a) identify and document the functional and physical characteristics of a configuration item, (b) control changes to those characteristics, and (c) record and report change processing and implementation status. (Reference MIL-STD-973)

Cycle/system test
The final phase of developer information systems testing which involves the testing of

modules/programs/cycles which are integrated into the total system.

Developer tests
Testing, modeling, and experimentation conducted by the system developer. Formal tests normally involve system level integration and certification by the developer with formal Government monitoring. Informal tests involve lower level code and unit development with internal integration between system elements. Experimentation includes a wide variety of tests, models, development techniques and simulations used to validate design concepts and theories.

Developmental tests (DT)
Tests usually conducted by an organization independent of the developer(s) in order to validate total system conformance to technical and functional specifications and ensure the system is ready for formal or limited user testing. Formal tests focus primarily on total systems integration.

Development tools
Products which are necessary to prepare, test and evaluate software units currently under development.

Driver
Software which controls a hardware device or the execution of other programs.

Dynamic analysis
A test method that involves executing or simulating a product under development. Errors are detected by analyzing the response of the product to sets of input data.

Emulation
An interpretation similar to simulation, however, the interpretation is done through hardware or microcode or the process of using software or peripherals to make one set of hardware operate like another.

Engineering change proposal - software (ECP-S)
A term which includes both a proposed engineering change and the documentation by which the change is described and suggested. DA Form 5005–R is used to document proposed changes to software baselines and associated baseline documentation. (Reference DA Pamphlet 25–6)

Firmware
A combination of hardware device and computer instructions or computer data that reside as read-only software on the hardware device. The software cannot be readily modified under program control.

Functional baseline
The initially approved documentation describing a system's or item's functional, interoperability, and interface characteristics and the verification required to demonstrate the achievement of those specified characteristics. (Reference MIL-STD–973)

Functional configuration audit (FCA)
A formal examination of the functional characteristics of a configuration item, prior to acceptance, to verify that the item has achieved the requirements specified in its functional and allocated configuration documentation. (Reference MIL-STD–973)

Hardware configuration item (HWCI)
A configuration item that is hardware. (Reference MIL-STD–973)

Implementation procedures (IP)
A document which provides information to users and data processing personnel to install the AIS and achieve operational status.

Independent verification and validation (IV&V)
Systematic evaluation performed by an agency that is not responsible for developing the product or performing the activity being evaluated. (Reference MIL-STD–973)

Interface
In software development, a relationship among two or more entities (such as CSCI-CSCI, CSCI-HWCI, CSCI-user, or software unit-software unit) in which the entities share, provide, or exchange data. (Reference MIL-STD–498)

Integrated product team (IPT)
A flexible and dynamic ad hoc group whose participants come from all necessary functional organizations in order to plan, manage, implement and resolve a particular acquisition program issue. (Reference DODD 5000.1, DOD 5000.2–R)

Interim change package (ICP)
A software modification release of ECP-Ss which, because of urgency, regulatory requirement or special need, must be provided before the availability of the next scheduled Software Change Package.

Interoperability
The ability of systems, units, or forces to provide services to and accept services from other systems, units or forces and to use the services to enable them to operate effectively together.

Issues and criteria
Issues are questions, the answers to which permit the overall system evaluation. Criteria are the quantitative or qualitative standards by which issues are evaluated.

Left-of-baseline (LOB)
The manual and automated processes of extracting selected data and reducing them to input file and transaction formats acceptable for building or initializing a database for a new system. Normally associated with conversion requirements or parallel testing.

Materiel system computer resources (MSCR)
Computer resources acquired for use as integral parts of weapons; command and control; communications; intelligence and other tactical or strategic systems and their support systems. The term also includes all computer resources associated with specific program developmental T&E, operational testing, and post deployment software support including weapon system training devices, automatic test equipment, land based test sites, and system integration and test environments.

Metric
A quantitative value, procedure, methodology, and/or technique which allows one the ability to measure various aspects and characteristics of software.

Nondevelopment item (NDI)
A generic term that covers material available from a variety of sources with little or no development effort required by the Government. NDI may be referred to as reusable, Government furnished, or commonly available software, hardware or total systems, depending upon the source. (Reference MIL-STD–973)

Parallel testing
Testing that demonstrates whether or not two versions of the same application are consistent, or two systems performing the same function.

Physical configuration audit (PCA)
The formal examination of the " as-built" configuration of a configuration item against its technical documentation to establish or verify the configuration item's product baseline. (Reference MIL-STD–973)

Program
A separately compilable, structural (closed) set of instructions most precisely associated with early generations of computers. Synonymous with computer program.

Qualification testing
Testing performed to demonstrate to the contracting agency that a CSCI or system meets its specified requirements. (Reference MIL-STD–498)

Recovery/reconfiguration testing
Testing that verifies the recovery process and component parts' effectiveness. It validates that enough backup data is preserved and stored in a secure location.

Regression testing
Testing of a computer program and/or system to assure correct performance after changes were made to code that previously performed correctly. Includes testing or retesting those areas or aspects of a system which will or could be affected by the changes.

Release
A configuration management action whereby

a particular version of software or documentation is complete and available for a specific purpose (e.g., released for test). (Reference MIL-STD–973)

Representative sample
For a program using an incremental acquisition strategy, the representative sample is the increments that will be used as the basis for a fielding decision. The chief characteristic of these increments is that they must constitute a self-sufficient package, i.e., they can stand alone.

Required operational characteristics
Qualitative and quantitative system performance parameters, proposed by the user and approved by the Army, that are primary indicators of a system's capability to accomplish its mission (operational effectiveness) and to be supported (operational suitability). Required operational characteristics are usually tested and evaluated by operational testing and evaluation to ascertain achievement of approved goals and thresholds for these characteristics.

Required technical characteristics
Quantitative system performance parameters approved by the Army management that are selected as primary indicators of technical achievement. These might not be direct measures of, but always should relate to a system's capability to perform its required mission function and to be supported. Required technical characteristics usually are tested and evaluated to ascertain approval goals and thresholds for these characteristics.

Requirements trace
Assuring requirements flow from the user specifications through design and implementation of the product.

Right-of-baseline (ROB)
The automated process of building a database from LOB products, or the initialization of new files introduced for the first time. Normally associated with conversion requirements or parallel testing.

Simulation
The process of conducting experiments with a model for the purpose of understanding the behavior of the system. Simulations may be dynamic, engineering (scientific), environmental, instruction level, statement level, and system level. For AIS, simulation entails summary files to simulate an internal or external interface input.

Software acceptance test (SAT)
A operational test of a new system or changes to a deployed system, directed by an independent tester and conducted in a field environment using a production database and executed on target hardware.

Software change package (SCP)
One or more changes which have been approved and scheduled for implementation, as a group, by the appropriate configuration control board IAW MIL-STD–973.

Software development
A set of activities that results in software products. Software development may include new development, modification, reuse, reengineering, maintenance, or any other activities that result in software products.

Software development file (SDF)
A repository for material pertinent to the development or support of a particular body of software. Contents typically include (either directly or by reference) considerations, rationale, and constraints related to requirements analysis, design, and implementation; developer internal test information; and schedule and status information.

Software development library (SDL)
A controlled collection of software, documentation, other intermediate and final software products, and associated tools and procedures used to facilitate the orderly development and subsequent support of software.

Software engineering environment (SEE)
The facilities, hardware, software, firmware, procedures, and documentation needed to perform software engineering. Elements may included, but are not limited to CASE tools, compilers, assemblers, linkers, loaders, operating systems, debuggers, simulators, emulators, documentation tools, and database management systems.

Software test environment
The facilities, hardware, software, firmware, procedures, and documentation needed to perform qualification, and possibly other, testing of software. Elements may include but are not limited to simulators, code analyzers, test case generators, and path analyzers, and may also include elements used in the software engineering environment.

Software qualification test (SQT)
Independent developmental test conducted on a target system, but normally not in an operational environment.

Software transition
The set of activities that enables responsibility for software development to pass from one organization, usually the organization that performs initial software development, to another, usually the organization that will perform software support.

Software unit
An element in the design of a CSCI; for example, a major subdivision of a CSCI, a component of that subdivision, a class, object, module, function, routine, or database. Software units may occur at different levels of a hierarchy and may consist of other software units. Software units in the design may or may not have a one-to-one relationship with the code and data entities (routines, procedures, database, data files, etc.) that implement them or with the computer files containing those entities. (Reference MIL-STD–498)

Statement of work (SOW)
A statement of contract requirements that is used for defining and achieving program goals. The SOW provides the basic framework for a particular effort. It is a document by which all nonspecification requirements for developer efforts must be established and defined either directly or with the use of specific cited documents.

Static analysis
A direct examination of the form and structure of a product without executing the product. It may be applied to requirements, design, or code.

Stress test
A test which exercises code up to, including and beyond all stated limits in order to exercise all aspects of the system (e.g., to include hardware, software, and communications). Its purpose is to insure that response times and storage capacities meet requirements.

Supplemental site test
Testing conducted on systems that execute in multiple hardware and operating system environments or for conditions/functions not readily available at a primary test site. (Reference AR 73–1)

Supportability
The degree to which a system can be maintained or sustained in an operational environment.

System change package
A group of modifications documented on ECPs which are packaged and implemented during post deployment phase.

System decision paper
The primary document used to obtain MAISRC approval for information systems. Also contains information comparable to the MSCR CRLCMP.

System post deployment review (SPR)
A review conducted after deployment of the initial system to evaluate how well the operational system is satisfying user requirements.

System specification
A system level requirements specification. A system specification may be a system/subsystem specification, prime item development specification, or critical item development specification). (Reference MIL-STD–498)

Target system
Suite of hardware, or hardware and software designated as the operational configuration of the system.

Test and evaluation master plan (TEMP)
The key management tool for control of the

integration of all T&E requirements for each acquisition effort and is used by decision review bodies. (Reference DODI 5000.1, DOD 5000.2–R)

Testbed
A system representation consisting partially of hardware and/or software and partially of computer models or prototype hardware and/or software.

Test hooks
Software logic integrated into a system to facilitate extraction of data to support test and analysis.

Test IPT
Established by the program sponsor upon receipt of the draft operational requirements document or mission needs statement. This is the primary group which facilitates integration of T&E requirements and prepares the TEMP.

Unit testing
The lowest level developer test of software.

User or operational tests
A system level test performed by a test activity independent of the developer or the PM. The objective of operational testing is to examine the entirety of the system and is performed by users in an operational environment.

Validation
The process of evaluating software to determine compliance with specified requirements.

Verification
The process of evaluating the products of a given software development activity to determine correctness and consistency with respect to the products and standards provided as an input to that activity.

Version
An identified and documented body of software. Modifications to a version of software (resulting in a new version) require configuration management actions by the developer, the Government or both. (Reference MIL-STD–973)

Walk-through
An informal, step-by-step review of a software product during development (i.e., program code, test scenario, functional design) which allows feedback from other members of the development team to the creator of the particular product being reviewed.

Section III
Special Abbreviations and Terms

This publication uses the following abbreviations, brevity codes, and acronyms not contained in AR 310–50.

Ao
operational availability

AAE
Army Acquisition Executive

ACAT
acquisition category

ACWP
actual cost of work performed

AIN
Army Interoperability Network

AIS
automated information system

AOI
additional operational issues

AQP
automation quality plan

ASARC
Army Systems Acquisition Review Council

ASDP
accelerated software development process

ATCOM
Aviation and Troop Command

ATE
automated test equipment

ATIRS
Army test incident reporting system

BCWP
budgeted cost of work performed

BCWS
budgeted cost of work scheduled

BMTF
benchmark test files

CASE
computer aided software engineering

CBTDEV
combat developer

CDRL
contract data requirements list

CEPT
concept evaluation program test

CMM
capability maturity model

CMU
Carnegie Mellon University

COIC
critical operational issues and criteria

COTS
commercial off-the-shelf

CPI
cost performance index

CPM
computer programming manual

CRISD
computer resources integrated support document

CRLCMP
computer resources life cycle management plan

CRMP
computer resources management plan

CRU
computer resource utilization

CRWG
computer resources working group

C/SCSC
cost/schedule control systems criteria

CSCI
computer software configuration item

CSE
Center for Software Engineering

CSOM
computer system operator's manual

CSU
computer software unit

DAA
designated accreditation authority

DAB
Defense Acquisition Board

DBDD
database design description

DEVLIB
development library

DIL
Digital Integration Laboratory

DISC4
Director of Information Systems for Command, Control, Communications, and Computers

DODISS
Department of Defense Index of Specifications and Standards

DTRR
developmental test readiness review

DTRS
developmental test readiness statement

ECP-S
engineering change proposal - software

EIA
Electronic Industries Association

EUE
early user experimentation

EUT
early user test

EUTE
early user test and experimentation

FAT
first-article test

FCA
functional configuration audit

FD
functional description

FDE
force development experiment

FD/SC
failure definition/scoring criteria

FDT
force development test

FOT
follow-on operational test

FP
functional proponent

FQT
formal qualification test

FSM
firmware support manual

GCCS
Global Command and Control System

HWCI
hardware configuration item

IAP
independent assessment plan

IAR
independent assessment report

IDD
interface design description, interface design document

IEC
International Electrotechnical Commission

IEEE
Institute of Electrical and Electronics Engineers

IMA
information mission area

IOT
initial operational test

IRS
interface requirements specification

ISC
Information Systems Command

ISEC
Information Systems Engineering Command

IV&V
independent verification and validation

JITC
Joint Integrated Test Center

JT
joint test

KPA
key process area

LCSEC
Life Cycle Software Engineering Center

LEA
Logistics Evaluation Agency

LOB
left-of-baseline

LUT
limited user test

MAIS
major automated information system

MAISRC
Major Automated Information System Review Council

MANPRINT
manpower and personnel integration

MATDEV
materiel developer

MCCR
mission critical computer resources

MDAP
major defense acquisition program

MEDCOM
Medical Command

MICOM
Missile Command

MNS
mission needs statement

MRRB
Materiel Release Review Board

MSCR
materiel system computer resources

NBS
National Bureau of Standards

NDI
nondevelopment item

OCD
operational concept description

OMS/MP
operational mode summary/mission profile

OPTEC
Operational Test and Evaluation Command

ORD
operational requirements document

OTP
outline test plan

OTRR
operational test readiness review

OTRS
operational test readiness statement

PDL
program design language

PDSS
post deployment software support

PPQT
pre-production qualification test

PR
problem report

QIO
Quality Improvement Office

ROB
right-of-baseline

RRR
RAM rationale report

SCMP
software configuration management plan

SCOM
software center operator manual

SDC-L
Software Development Center - Lee

SDC-W
Software Development Center - Washington

SDL
software development library

SDT
software development test

SEE
software engineering environment

SEI
Software Engineering Institute

SER
system evaluation report

SIOM
software input/output manual

SLOC
source line(s) of code

SMERFS
Statistical Modeling and Estimation of Reliability Functions for Software

SMMP
system MANPRINT management plan

SPCR
software problem/change report

SPI
schedule performance index

SQA
software quality assurance

SQPP
software quality program plan

SRTM
software requirements traceability matrix

SRS
software requirements specification

SSDD
system/segment design document, system/subsystem design description

SST
supplemental site test

STD
software test description standard

STP
software test plan

STR
software test report, software trouble report

STrP
software transition plan

SVD
software version description

S/W
software

TAFIM
technical architecture framework for information management

T&E
test and evaluation

TEMP
test and evaluation master plan

TEXCOM
Test and Experimentation Command

TIR
test incident report

TQM
total quality management

UAT
user acceptance test

UM
user manual

US
software unit specification

VDD
version description document

V&V
verification and validation

WIPT
working-level integrated product team

Index

This index is organized alphabetically by topic and subtopic. Topics and subtopics are identifed by paragraph number and appendix (when appropriate).

USAPA

ELECTRONIC PUBLISHING SYSTEM
TEXT FORMATTER ... Version 2.61

PIN:	075103–000
DATE:	06-21-99
TIME:	15:45:43
PAGES SET:	108
DATA FILE:	p73.fil
DOCUMENT:	DA PAM 73–7
DOC STATUS:	NEW PUBLICATION